Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vytvoření plug-inu pro virtuální hudební studio

VST Plug-In for Digital Audio Workstation

2014 **Bc. Antonín Šleis**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Antonín Šleis**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Vytvoření VST plug-inu pro virtualní hudební studio
VST Plug-In for Digital Audio Workstation

Zásady pro vypracování:

Cílem práce je popsat standard VST pro pluginy virtuálních hudebních studií a vytvořit vlastní funkční plugin.

1. Popis VST standardu.
2. Návrh grafického rozhraní s popisem jednotlivých prvků.
3. Implementace VST pluginu. Plugin bude sloužit jako modulační efektová jednotka a bude umožňovat např. výběr zdrojů k modulaci a přiřazení modulátorů. Pro vývoj bude použito např. SDK fy Steinberg.
4. Otestování plug-inu na platformě virtuálního hudebního studia.

Seznam doporučené odborné literatury:

[1] Mike Collins. *A Professional Guide to Audio Plug-ins and Virtual Instruments*. Elsevier, 2003. Počet stran: 656.
[2] Richard Grace. *Hudba a zvuk na počítači*. Grada, 1998. ISBN: 807169519X.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Skapa, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: *7. května 2014*

podpis studenta

Poděkování

Rád bych poděkoval Janu Skapovi Ph.D za odbornou pomoc a konzultaci při vytváření této diplomové práce.

Abstrakt

Tato diplomová práce popisuje vývoj zvukové techniky a její částečnou migraci ze světa fyzických zařízení na platformu počítačů pomocí simulace původních zařízení. Detailně rozebírá technologii Virtual Station Technology (VST) firmy Steinberg, sloužící k integraci VST zvukových zásuvných modulů do novodobých softwarových digitálních zvukových stanic (DAW). Následná implementace inovativního VST zásuvného modulu 3TMatrixPlugin demonstruje možnosti této technologie. Užití poměrně nového, krom původní dokumentace dosud nepopsaného frameworku VST.NET, poskytuje úvod do modernějších a efektivnějších metod vývoje, než nabízí původní Steinberg VST SDK napsaný v nespravované verzi jazyka C++. Dalším cílem zpracování tématu je představení málo známé platformy pro aplikaci algoritmů probíraných v rámci předmětu Digitální zpracování signálů na naší univerzitě.

Klíčová slova

Steinberg VST SDK; Digitální zpracování signálu; DAW; VST.NET; Zvuková technika;

Abstract

This thesis describes the development of sound technology and its partial migration from the world of physical devices to the platform of the computer using simulation of the original devices. Closely analyzes the technology of Steinberg Virtual Station Technology (VST), serving to integrate VST audio plug-ins in modern digital audio workstations (DAW). The subsequent implementation of innovative VST plug-in 3TMatrixPlugin demonstrates the power of this technology. The use of the relatively new VST.NET framework, which has not been documented except for the original documentation, provides an introduction to modern and effective methods of development not offered by the original VST plugin SDK written in an unmanaged version of the C++ language. In addition the objective of the thesis is to introduce a little-known platform for the application of the algorithms discussed in the course Digital Signal Processing at our university.

Key words

Steinberg VST SDK; Digital signal processing; DAW; VST.NET; Sound technology;

Seznam použitých zkratek

Zkratka	Význam
VST	Virtual Station Technology
DAW	Digital Audio Workstation
MIDI	Musical Instrument Digital Interface
DSP	Digital Signal Processing
LFO	Low Frequency Oscillation
SDK	Software Development Kit
WYSIWYG	What you see is what you get
API	Application Programming Interface

Seznam použitých termínů

Termín	Význam termínu
Preset	Převolba
Delay	Echo (ozvěna)
Plugin	Zásuvný modul
Sample	Zvukový vzorek
Warmer	Druh zvukového efektu řešící „studenou“ povahu digitálních zvuků
Efektový rack	Studiová skříň (v PC panel) pro umístění zvukových efektů
Knob	Otočný potenciometr
Opcodes	Část strojových instrukcí počítače
Interoperabilita	Schopnost různých systémů vzájemně spolupracovat
Hi-End	Zboží nejlepší kvality (nejvyšší ceny)

Obsah

Úvod

Počátky vnímání sekvencí úderů jako hudby sahají až do pravěku. Lidé zjistili, že rytmický doprovod při vykonávaní určité činnosti zvyšuje produktivitu práce a synchronizuje jednotlivé zúčastněné. Začali také obohacovat rituály u táborových ohňů periodickými údery pro navození stavů duševního transu. Postupně své techniky vylepšovali a již 5000 let př.n.l. byla hudba součástí všeobecného vzdělání. Za pomoci pouhého sluchu lidé vypozorovali neuvěřitelně přesně zákonitosti, jako jsou pentatonika nebo rozdělení zpěvných hlasů, které byly vysvětleny fyzikou až o mnoho stovek let později. Již tehdy si uvědomovali úzké propojení hudby a matematiky. Hudba se dál rozvíjela, zdokonalovala a umožňovala výrobu stále složitějších skladeb. Také technologie výroby hudebních nástrojů šly kupředu. Nicméně byly všechny postaveny na akustických vlastnostech použitých materiálů. Až do nedávného objevení elektřiny, kdy hudba společně s celým světem začala procházet revolucí takových rozměrů jako nikdy dříve…

Jakožto dlouholetého milovníka hudby mě přirozeně časem začalo zajímat, jak hudba „funguje“. Usoudil jsem, že nejlepším výchozím bodem bude studium hry na klavír. Ten zdaleka nepatří k nejjednodušším hudebním nástrojům, ale díky své komplexnosti umožňuje proniknutí do tajů hudební teorie nejvíce. Po několika letech studií notového zápisu, techniky hraní a hudební teorie jsem navštívil kamaráda, který se již nějakou dobu věnoval tvorbě vlastních skladeb. Zde jsem poprvé uviděl domácí hudební studio v čele s počítačem pro míchání zvuků pocházejících z různých nástrojů. Vypůjčil jsem si program k tomu určený Fruity-loops 3 a nainstaloval doma. Byl jsem ohromen propojením světa akustické hudby a světa výpočetních technologií. Začal jsem tedy se softwarem experimentovat a po nějaké době také vyprodukoval první skladbu. V té době jsem již nastoupil na univerzitní studia informatiky a tak mě pochopitelně začalo zajímat, jak celý tento proces probíhá z technického úhlu pohledu. Jak to celé pracuje a jak by se dal případně takový program vytvořit. V posledním roce bakalářského studia jsem uvažoval o zvolení této problematiky jako tématu mé bakalářské práce. Bohužel jsem zjistil, že informací o patřičných technologiích je jak v literatuře tak na internetu velice málo. Také implementace není přímočará a je třeba znát mnoho věcí kolem. Po zvážení náročnosti a rozsahu jsem zvolil téma jiné a volnočasovým studiem se postupně připravoval tak, abych tuto problematiku mohl zpracovat v rámci práce diplomové.

Tato práce popisuje základní procesy ve zvukové technice a možnosti implementace vlastních zvukových zásuvných modulů v rámci standardu VST. Konečným produktem je VST zásuvný modul 3TMatrixPlugin, který je možno použít jako modulační efektovou jednotku ve všech nejznámějších digitálních zvukových stanicích na platformě Windows. Například stanice Cubase firmy Steinberg, FL Studio firmy Image-Line Software nebo Ableton Live firmy Ableton.

Účel práce je mimo naprogramování inovativního VST zásuvného modulu, také zdokumentování této sofistikované části informatiky a tím poskytnutí základních informací pro

studenty naší univerzity zajímající se o programování zvuku. Na tuto práci budou moci navázat a investovat svůj čas do tvorby pokročilejších zvukových modulů.

Text je členěný do logických celků. V první kapitole rozebírám problematiku zvukové techniky od jejích počátků začátkem 20. století až po současnost. Popisuji výhody a nevýhody počítačové simulace zvukových zařízení. Druhá kapitola se věnuje detailnímu rozboru technologie VST a jednotlivých typů zvukových přístrojů. V této části práce také uvádím dostupné prostředky k vývoji VST programů. Kapitola čtyři popisuje návrh zásuvného efektového modulu VST, z kterého vychází následná implementace a nakonec testování dosažených výsledků. V závěru se můžete dočíst o mé vizi do budoucna, pokračování této práce a jejím přínosu.

V této práci rozděluji nástroje zvukové techniky na hardwarové a softwarové. Hardware se obecně chápe jako komponenta počítače. Já si toto označení vypůjčil pro odlišení fyzických zařízení a těch počítačových. Přístroje zvukové techniky často slouží jako komponenty komplexnějších systémů například hudebních studií. Tudíž označení "hardwarový" zde znamená hardwarové zvukové komponenty zvukových studií a ne komponenty počítače (počítačový hardware).

Jako poslední bych chtěl v úvodu říci, že překlad některých všeobecně zažitých termínů do češtiny by dle mého názoru vedl pouze k zmatení čtenáře. Proto je tedy nechávám v původní podobě a jejich vysvětlení uvádím v seznamu použitých termínu.

1 Analýza problému moderních zvukových aplikací

1.1 Historie elektronické hudby

Počátkem 20. století bylo lidstvo, díky objevení elektřiny a moderní fyzice, která vysvětluje zvuk jako akustické vlny, již jen malý krůček od výroby prvního elektronického hudebního nástroje. Hudebníci té doby uvažovali, jak uchopit moderní hudbu. Běžné instrumentální orchestry s klasickou funkční harmonií začaly být svazující a neuspokojovaly novou generaci hudebníků prahnoucích po odlišnosti jejich práce. Zástupci druhé vídeňské školy Arnold Schönberg, Anton Webern a Alban Berg rozbili tónové struktury a začali skládat podle nové koncepce dodekafonie. Ve stejné době svět zvuku zažíval obrovský rozmach nových technologií, jako jsou telefon, gramofon a mikrofon. Ty umožňovaly poprvé zaznamenávat zvuky. Netrvalo dlouho, než si hudebníci tyto přístroje osvojili a začali je používat pro zakomponování nezvyklých zvuků, hluků a ruchů do svých skladeb. Například italský futurista Luigi Russolo byl fascinován městským hlukem a v roce 1913 publikoval knihu *L'arte dei rumori L'arte dei rumori* (Umění hluku), kde propaguje odklon od klasických tónů směrem k hlukům.

Vůbec prvním opravdovým elektronickým nástrojem byl teremín, který vynalezl v roce 1919 ruský inženýr Lev Těrmen (v zahraničí znám jako Leon Theremin). Zvláštnost je, že se na tento nástroj hraje bezdotykově. Ruce pouze krouží kolem antény, čímž se mění výška a síla zvuku. Poprvé teremín použil skladatel Paul Hindemith ve skladbě *Educatorial*. Poté v roce 1930 uvedl jeho žák Oskar Sala nový elektronický nástroj trautonium. Ten se již koncepčně velmi blížil moderním elektronickým klávesám.

V roce 1939 velice populární experimentální skladatel Johny Cage použil pro svou skladbu *Imaginary Landscape No 1* kromě akustických nástrojů také gramofony s elektronickými zvuky různé frekvence. Obrovským krokem vpřed bylo vynalezení magnetofonové pásky v roce 1932. To umožňovalo stříhání a slepování jednotlivých zvukových pasáží. Toho využili v roce 1948 Pierre Schaeffer a skladatel Pierre Henry k složení sbírky etud *Études de bruits* s použitím všednodenních zvuků. Tento impuls byl pro rozvoj elektronické hudby zásadní, neboť se skladatel stává zároveň interpretem. Nově poskládal zaznamenané kousky zvuku, což je dnes běžnou činností producentů elektronické hudby.[1]

Dalším významným průkopníkem elektronické hudby byl skladatel a muzikolog Herbert Elimert z kolínské televizní a rozhlasové stanice WDR. Vytvořil nové studio elektronické hudby, které se velice rychle stalo místem setkávání skladatelů z celého světa. Nejznámější člen tohoto studia Kalheinz Stockhausen uvedl v padesátých letech spoustu hudebních děl, kde využíval magnetonové pásky a čisté sinusové tóny (bez vyšších harmonických tónů).

Zvuková technika byla pro rozšíření napříč světem stále příliš velká a drahá. Až v šedesátých a sedmdesátých letech se díky technologickému pokroku začala vyrábět menší a levnější. Stala se dostupnější pro širší veřejnost. Mnoho rockových kapel, jako jsou například

Silver Apples nebo Pink Floyd, si velice rychle elektronickou zvukovou techniku osvojilo. Tyto přístroje umožnovaly vyrobit naprosto jedinečný zvuk, který jim pomohl vyniknout. Nejvýznamnějším projektem této doby byla německá kapela Kraftwerk, která postavila svou hudbu výhradně na elektronických zvucích, syntetizérech a robotických modulacích zvuku.

Během několika let se objevilo mnoho významných a dodnes známých hudebníků vystupujících s elektronickými hudebními nástroji, jako jsou Brian Eno, Vangelis, Jean Michel Jarre nebo Isao Tomita. Nicméně pro masové rozšíření a definitivní vzestup elektronické hudby jak ji známe dnes, byl stěžejní filmový průmysl. Jedním z prvních režisérů, který jí použil, byl Stanley Kubrick. Jeho filmový hity Mechanický pomeranč se brzy rozšířil po celém světě a společně s ním i elektronická hudba.[2]

Prvním elektronickým hudebním žánrem byl Industrial music, který je postaven na těžkých elektronických industriálních zvucích. Příchod elektronického aranže skladeb poskytnul větší přesnost, rychlost a složitost bicích, než umožňuje klasická hra na perkuse. Navíc klasické zvuky bicích mohly být nahrazeny jejich elektronickými ekvivalenty (často upravenými k nepoznání). To mělo v 80. letech za následek vznik mnoha elektronických tanečních žánrů. Například Techno music v Detroitu, House music v Chicagu a později také Acid House music ve Velké Británií.

Koncem 80. let zažíval velký rozmach osobní počítač. Pochopitelně se začalo uvažovat o jeho využití k vytváření elektronických zvuků a nahrávek. Během následujícího desetiletí se díky pokroku techniky, opětovnému snižování cen, velikosti nástrojů a zvyšování výkonu osobních počítačů kompletně změnil koncept obrovských hardwarových extrémně drahých studií na malé hudební pracovní stanice. To zpřístupnilo využití elektronické hudební techniky téměř komukoliv. Po celém světě začaly vznikat desítky nových progresivních hudebních stylů a také nesčetné množství inovativních technik tvorby hudby.

1.2 Digitální zvuková pracovní stanice (Digital Audio Workstation)

Je elektronický systém sloužící výhradně nebo primárně k nahrávání, upravování a přehrávání digitálního zvuku.[3]

1.2.1 První generace digitálních zvukových pracovních stanic

První generace digitálních zvukových pracovních stanic se vyznačovala levným mikroprocesorem, který řídil chod celého přístroje (správu předvoleb atd.). Dále poskytovala diskové úložiště pro zvukové vzorky a rozhraní k připojení kláves. V novějších modelech se začínala objevovat kromě FM syntézy a samplování také syntéza digitální. Jednou z prvních byla stanice Synclavier 1 vyrobena v roce 1977, firmou New England Digital Corporation (obrázek 1.1). Tento komplet využíval FM syntézu a postrádal klaviaturu. Další verze Synclavier 2 rozšířila Synclavier 1 o On/Of klaviaturu a umožňovala 16-bitové zpracování zvuku při nejvyšší vzorkovací frekvenci 100kHz.

Obrázek 1.1: *Jedna z prvních digitálních hudebních zvukových stanic Synclavier1 [3]*

1.2.2 Druhá generace digitálních zvukových pracovních stanic

Hudební stanice postupně vylepšovaly své možnosti, například začaly poskytovat vyšší pracovní vzorkovací frekvenci nebo přechod z 8-bitového zvuku na 16bitový. Největší změna však přišla v roce 1983 díky novému protokolu MIDI. Šlo o nový standard používaný v hudebním průmyslu sloužící jako komunikační protokol mezi přístroji (bližší informace zde [4]). Také se začaly objevovat příjemné grafické uživatelské rozhraní pro ovládání zvukových pracovních stanic. Levné stanice nabízely led diodové displeje, ale ty dražší již poskytovaly uživateli interakci skrze monitor osobního počítače. Častým se také stalo integrování digitálních efektů dovnitř nástrojů namísto využití externích modulů.

Na obrázku 1.2 je zástupce druhé generace zvukových pracovních stanic Korg M1, který byl představen v roce 1988. Během několika let se prodalo nad 250 000 kusů, což z něj dělá jednu z nejúspěšnějších hudebních stanic vůbec. Umožňoval hru až šestnácti různých tónů najednou. Díky kombinaci digitální samplované syntézy a následnému obohacení zvuku o analogovou složku generovanou z přístroje vydával opravdu bohatý, barevný a přirozený zvuk. Dále obsahoval mnoho prvků pro úpravu a modulaci základního zvuku, jako jsou nízkofrekvenční oscilátory (LFO) nebo obálky ADSR. Není divu, že jej používalo mnoho populárních hudebníků. Například kapela Queen, Madonna, Depeche Mode a spousty dalších. Více informací o legendě Korg M1 naleznete zde [5], odkud také pochází obrázek 1.2.

Obrázek 1.2: *Hudební pracovní stanice druhé generace Korg M1*

1.2.3 Moderní hudební pracovní stanice

S pokrokem technologií se stále rozvíjely možnosti těchto hudebních stanic a dnes můžeme na trhu najít opravdu komplexní nástroje s velkými dotykovými displeji, obrovským množstvím předvoleb, zvukových efektů a možností propojení s dalšími zařízeními. Příkladem může být technologicky vyspělý model Korg Kronos na obrázku 1.3. Již na první pohled lze vidět pokročilou výbavu proti jeho předchůdcům.

Obrázek 1.3: *Moderní hudební pracovní stanice Korg Kronos*

1.2.4 Počítačová simulace hudebních zvukových pracovních stanic

Počítačová simulace hardwarů se úzce prolíná s pojmem virtualizace a dokonce mnohdy znamenají totéž. Virtualizace je v informatice označení postupů a technik, které umožňují v počítači přistupovat k dostupným zdrojům jiným způsobem, než jakým fyzicky existují, jsou propojeny atd. Virtualizované prostředí může být mnohem lépe přizpůsobeno potřebám uživatelů, snáze se používat, případně před uživateli zakrývat pro ně nepodstatné detaily (jako např. rozmístění hardwarových prostředků). [6]

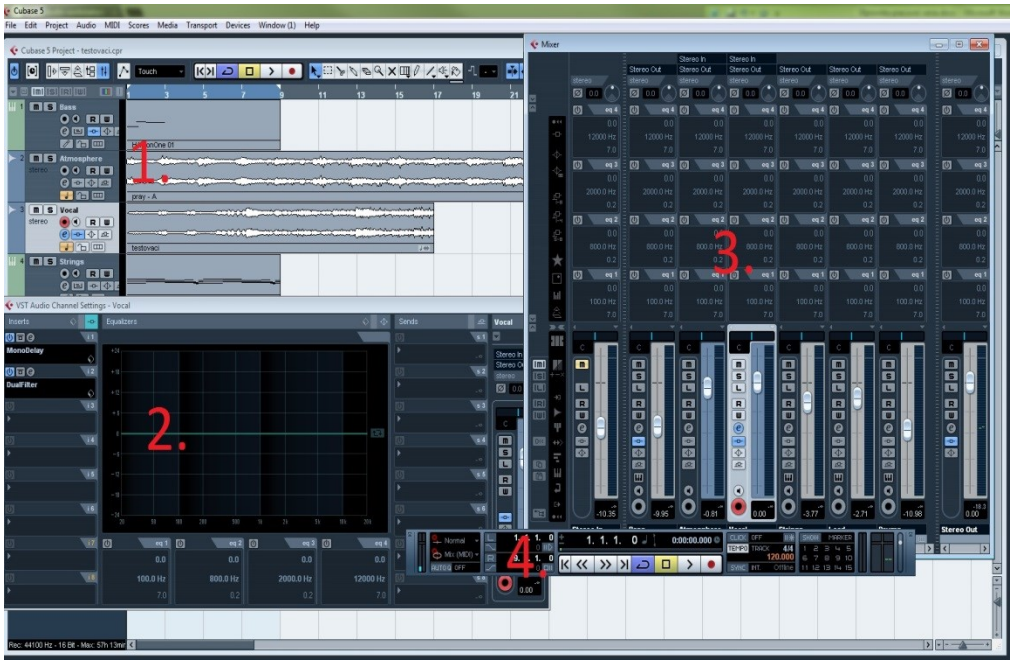
Tento proces je dnes všudypřítomný. Téměř vše z fyzického světa, již bylo více či méně nasimulováno v počítači. Ve světě hudební techniky tomu není jinak. S rostoucím výkonem počítačů roste také věrohodnost simulovaných přístrojů. V současnosti lze říci, že je možno věrohodně zvirtualizovat (simulovat) prakticky všechny přístroje zvukové techniky. Vzniknuvší počítačové programy navíc mohou lehce překonat mnoho omezení fyzických přístrojů, jako jsou například vysoká cena potřebných komponent nebo konstrukční složitost. Proto jdou často zvukové programy mnohem dále, než by byly schopny jejich fyzické ekvivalenty.

Na druhou stranu je třeba také zmínit mínusy. Prvním z nich je fakt, že u některých simulovaných zvukových zařízení není barva a „teplo“ výsledného zvuku zcela totožná se zvukem zpracovaným analogovými hardwarovými přístroji. V některých případech působí výsledný zvuk „digitálně studeně“. Problém způsobuje hlavně rozdíl mezi spojitým analogovým signálem používaným ve fyzických přístrojích a jeho nedokonalou digitální reprezentaci v počítači. Jak tento rozdíl co nejlépe překonat, je stále otevřené téma a mnoho výrobců zvukových softwarů začalo vyrábět takzvané Warmery, jenž lépe či hůře zmíněný problém řeší. Druhým nedostatkem je pořád znatelná limitace zvukových programů výkonem počítače.

1.2.5 **Softwarové digitální zvukové pracovní stanice**

Díky virtualizaci a zlepšování výkonu počítačů se začátkem devadesátých let začaly objevovat hybridy zvukových pracovních stanic, které kombinovaly hardwarové digitální DSP komponenty s počítačovým rozhraním pro jejich obsluhu. Vůbec první zvuková pracovní stanice založena čistě na software byla uvedena v roce 1993. Jednalo se o Samplitude Studio firmy SEK‘d. V roce 1996 uvedla německá firma Steinberg svou čistě softwarovou verzi zvukové pracovní stanice Cubase VST 3.0, která kromě nahrávání a upravování zvukových stop také poskytla zvirtualizovanou verzi mixážního pultu a efektového racku. Od této chvíle softwarová zvuková pracovní stanice obsahovala vše, co nabízela velká analogová studia. Tato koncepce byla brzy převzata ostatními výrobci a přetrvává dodnes.

Grafické rozhraní softwarové digitální zvukové stanice může být implementováno jakkoli, avšak všeobecně se výrobci snaží držet konceptu založenému na Multitrack recordingu (systém pro záznam více zvukových stop najednou). Díky tomu mohou zvukoví inženýři, v minulosti pracující se systémem Multitrack recording, snadněji přejít na tyto softwarové stanice. Rozložení grafických prvků softwarových stanic obvykle obsahuje panel pro ovládání přehrávání a nahrávání zvuku, mixér, ovládací panel jednotlivých stop s efektovým rackem a sekci pro zobrazení zvukových vln jednotlivých stop. Na obrázku 1.4 je grafické rozhraní softwarové hudební stanice Cubase 5, kterou budu používat pro testování výsledného zvukového zásuvného modulu. Porovnat jí můžete s profesionálním hardwarovým hudebním studiem na obrázku 1.5. Jeho pořizovací cena se pohybuje v řádech miliónů korun českých. Podobně vybavené softwarové studio se dá pořídit za statisíce. Je třeba uvést, že srovnání není zdaleka tak přímočaré a slouží zde pouze k demonstraci výhod softwarového řešení.



Obrázek 1.4: *Softwarová zvuková stanice Cubase 5. 1. Sekce pro vizualizaci zvukových vln. 2. Ovládací panel jednotlivých stop s efektovým rackem 3. Mixér 4. panel pro ovládání přehrávání a nahrávání zvuku*

Obrázek 1.5: *Profesionální hudební studio založené na hardwarových komponentách. 1. Mixér, 2. Efektový rack*

1.2.6 **Zvukový zásuvný modul (Audio Plug-In)**

Zásuvný modul je obecně softwarová komponenta, která je přidána do aplikace koncovým uživatelem, za účelem rozšíření funkčnosti původního programu. Například internetový prohlížeč využívá zásuvné moduly pro rozšíření funkcionality o překlad zobrazených stránek nebo pro blokování nevyžádaných reklam. Výrobci aplikací obvykle vystavují externím programátorům aplikační rozhraní zvané API pro vývoj vlastních zásuvných modulů. Počáteční verze softwaru tedy může být šířena jen se základními prvky, které si pak může uživatel libovolně rozšířit. V DAW se většinou jedná o zásuvný modul simulující hudební nástroj nebo efektovou jednotku pro zpracování zvuku.

Jestliže chci naprogramovat vlastní zvukový modul, potřebuji získat API od výrobce, seznámit se s komunikací hostovací aplikace se svými moduly a znát formát, v jakém hostovací aplikace moduly očekává. Toto všechno se dá chápat jako standard, kterých existuje celá řada. Kdysi byl oblíbený standard DirectX. Ten lze ale použít výhradně pod operačním systémem Windows. Navíc se u něj vyskytovaly problémy s latencí a automatizací parametrů, proto se již dnes pro psaní zvukových zásuvných modulů příliš nepoužívá. Dalším standardem je RTAS (Real-Time AudioSuite) firmy DigiDesign (dnes Avid Technology). Podporuje ho výhradně zvuková pracovní stanice Pro Tools, která je primárně určena pro Mac počítače. Bohužel tak jako ve všech ostatních oblastech programování pro Mac i zde naráží entuziasmus programátorů na množství překážek. Jestliže chcete pomoci RTAS vyvíjet vlastní zásuvný modul, je potřeba kontaktovat firmu Avid a sjednat oficiální výrobu RTAS modulu, což je neslučitelné s open-source myšlenkou. Pro Mac počítače je také velice oblíbeným standardem Audio Units (AU), který je součástí operačního systému. To poskytuje nízkou latenci a stabilitu.

Nejuniverzálnějším a nejlépe dostupným standardem je VST(Virtual Station Technology) firmy Steinberg. Výhodou oproti AU je nezávislost na operačním systému. Dnes prakticky nenajdete pokročilejší zvukovou pracovní stanici, která by tento standard nepodporovala. Proto jsem pro můj zvukový zásuvný modul vybral právě tuto technologii, které dále věnuji celou kapitolu.

2 Virtual Studio Technology (VST)

VST technologie je rozhraní pro integraci audio instrumentů a audio efektů do DAW nebo zvukových editorů. VST a podobné technologie využívají digitálního zpracování signálu k softwarové simulaci tradičních hardwarů nahrávacích studii. Definice VST přeložená z anglické Wikipedie [7].

Oficiální dokumentace Steinberg VST SDK 2.4 říká, že VST zásuvný modul je pouze zvuk zpracovávající komponenta a ne zvuková aplikace. Je to komponenta využívaná v prostředí hostovací aplikace, která poskytuje VST modulu zvukové data k zpracování. Obecně řečeno, VST zásuvný modul může vzít zvukové data, zpracovat je a vrátit výsledek hostovací aplikaci. VST modul může zpracovávat zvuková data pomocí procesoru počítače. Není tedy třeba speciálního DSP procesoru. Proud zvukových dat je rozdělen do sérií bloků. Hostovací aplikace poskytuje tyto bloky modulům a stará se o správu momentální velikosti bloku (Block-size). VST zvukový modul udržuje informaci o stavu všech jeho parametrů, zapojených do zpracování zvuku. Z pohledu hostovací aplikace je VST zásuvný modul černá skřínka s libovolným počtem vstupů, výstupů a parametrů. Parametry mohou byt libovolně použity k ovlivnění interních procesů zpracování zvuku ve VST modulech. Podle možností hostovacích aplikací mohou být jednotlivé změny parametrů automatizovány.[8]

Specifikaci VST rozhraní a SDK uvolnila německá firma Steinberg v roce 1996 v rámci nové verze digitální softwarové pracovní stanice Cubase 3.0. Ta obsahovala několik zásuvných efektových modulů, jako jsou reverb (viz podkapitola 2.2.2), chorus (kapitola) nebo Auto-Panner.

Steinberg v roce 1999 vylepšil původní specifikaci VST na verzi 2.0. Nejvýznamnější inovací byla možnost zásuvných modulů přijímat MIDI události. Více se můžete dočíst o MIDI protokolu zde [4]. To podpořilo vznik VSTi (VST instrument) formátu, který umožňuje VST modulům fungovat nejen jako přídavné efektové jednotky ale také jako nezávislé zdroje zvuku. V roce 2006 Steinberg vydal dnes nejvíce podporovanou a používanou verzi VST 2.4. V roce 2008 vyšla verze VST 3.0 a pak v roce 2011 verze 3.5. Ty se však podle mého průzkumu netěší velké oblibě programátorů. Hlavně díky velké odlišnosti struktury SDK proti VST 2.4 je velice problematické migrovat starší VST moduly do nové verze. Také ne všechny DAW se dobře vypořádaly s podporou novějších verzí. Největší podporu a zdroje informací naleznete pro verzi 2.4. Bohužel Steinberg během psaní mé práce přestal oficiálně VST SDK 2.4 distribuovat. Tím chce pravděpodobně popohnat vývojáře k rychlejší migraci na novější verze.

Základní druhy VST zásuvných modulů simulují hardwarové přístroje používané ve zvukové technice. V této kapitole popisuji ty nejznámější z nich. Společně uvádím názorný hardwarový příklad a k němu příslušný softwarový VST ekvivalent.

Vzhledem k požadované funkcionalitě VST zásuvných modulů se dělí na tři základní skupiny:

- VSTi - VST instrumenty
- VST efekty - efekty zpracovávající zvukový signál
- VST MIDI efekty - efekty pracující s MIDI protokolem

Ačkoli jsou tyto programy napsány pod stejným standardem, každá skupina vykonává zcela odlišnou práci. Uvědomit si, do které skupiny vyvíjený modul spadá, je základním předpokladem k úspěšné implementaci.

VST MIDI efekty tvoří specifickou skupinu efektů, jejíž popsání by vydalo na další takovou práci. Protože se tak úplně netýkají mnou vyvíjeného modulu, záměrně je přeskočím. Případné zájemce o tématiku MIDI zde odkazuji na českou „bibli“ světa MIDI „Velká kniha MIDI“, kde najdou mnoho užitečných informací.

2.1 VSTi - VST instrumenty

Obdobně jako ve fyzickém světě, kde hudebním nástrojem (instrumentem) může být v podstatě cokoliv, co vydává zvuk, jsou ve virtuálním světě VST standardu právě programy generující nějaký zvukový signál považovány za VST instrumenty.

Podle povahy zdroje takového signálu je můžeme rozdělit do dvou kategorií:

- Samplery
- Syntetizéry

2.1.1 Samplery (sample player)

Z názvu je patrné, že se jedná o přehrávač samplů (zvukových vzorků). Sampler je tedy hudební nástroj umožňující přehrání zvuků, které byly nahrány uživatelem např. mikrofonem nebo načteny z nějaké předem vytvořené zvukové banky. Takový sampler má obvykle několik kláves nebo tlačítek, prostřednictvím kterých je schopen poslat na výstup patřičný zvuk (sample). Hardwarové samplery obvykle reagují na stisk tlačítka člověkem, ty softwarové pak na informaci MIDI nebo kliknutí uživatele na konkrétní tlačítko. Vlevo na obrázku 2.1 můžete vidět dnes velice oblíbený hardwarový sampler k vytváření beatů MPC2500 od firmy AKAI a pro srovnání vpravo softwarový VSTi zásuvný modul od firmy Steinberg Groove agent one.

Obrázek 2.1: *Samplery vlevo AKAI MPC2500 a vpravo Grove agent one*

2.1.2 **Syntetizéry**

Jsou hudební nástroje tvořící výsledný zvuk pomocí syntézy. Podle formy zpracovávaných zvuků jsou syntetizéry děleny na digitální a analogové. Podle formy existence syntetizéru dále dělí na hardwarové a softwarové. Hardwarové syntetizéry jsou v mnoha případech vybaveny klaviaturou, ale nemusí tomu tak být vždy. Mohou disponovat i jiným ovladačem, případně být použity jako moduly do větších systémů. Softwarové syntetizéry jsou programy generující signál v prostředí počítače. Signál z hardwarového nebo softwarového syntetizéru je pak k dispozici pro zpracování různými efekty nebo rovnou poslán na výstup k přehrání. Klasickým výstupem je obvykle reprobedna nebo sluchátka. Analogicky je VSTi syntetizér program napsaný ve standardu technologie VST poskytující zvukový signál pro pozdější zpracování v prostředí DAW. Příklad legendárního hardwarového syntetizéru Korg ms20 je vlevo na obrázku 2.2. Vpravo můžete vidět notorický známý softwarový VSTi syntetizér Massive od firmy Native instrument. Ten má velký úspěch hlavně díky nestandardním inovativním přístupům k vytváření a tvarování zvuků, jimž mohou hardwarové syntetizéry jen stěží konkurovat. Zde je obzvláště dobře vidět potenciál VST technologie.

Obrázek 2.2: *Vlevo hardwarový syntetizér Korg ms20 a vpravo softwarový VSTi syntetizér Massive*

2.2 VST efekty

Zvukový efekt je zvukový procesor pro úpravu zvukového signálů. Typicky přijme na svém vstupu signál, ten nějakým způsobem zpracuje a poskytne na výstupu. Tyto efekty jsou aplikovány všude kolem nás, aniž bychom si to uvědomovali. Proč jsou reklamy v televizi hlasitější než film? Ve skutečností hlasitější nejsou, jenom pro upoutání naší pozornosti, chytře využívají schopnosti zvukového efektů kompresoru. Proč zní zpěváci v hudebních soutěžích někdy až směšně nepřirozeně? Protože na jejich hlasy audio inženýři aplikují zvukový efekt Auto-Tune, který umí manipulovat s výškou tónů. Jsou tedy schopni ladit zpívané tóny a tím z dobrých zpěváků udělat ještě lepší nebo ze špatných ještě horší (samozřejmě kvůli sledovanosti).

Pro úplnost stojí za zmínku speciální skupina zvukových efektů - spektrální analyzéry, stereo analyzéry a celá řada pomocných měřidel zvukových signálů. Zvláštností u nich je, že tyto moduly nijak zvuk neupravují, pouze slouží jako vizualizační pomůcka zvukových inženýrů.

Dnešní mixování hudebních nahrávek se prakticky neobejde bez použití zvukových efektů. Ty jsou v některých případech aplikovány jen opravdu lehce. Například pro upravení dynamiky bicích. Někdy však skladbu ovlivňují ve velkém měřítku a po jejich odstranění byste skladbu ani nepoznali. Některé hudební žánry jsou dokonce na použití efektů zcela založeny. Styly jako jsou metal, drum and bass, electro house, trance a mnoho dalších by bez zařazení zvukových efektů nikdy nevznikly. V minulosti bylo velice těžké nějaký zvukový efekt na nahrávanou skladbu aplikovat. Například všeobecně známá ozvěna se v minulosti simulovala pomocí takzvaných dozvukových komor. Zpěvák zpíval do mikrofonu, ze kterého šel signál do mixážního pultu. Tam se duplikoval a jedna kopie byla poslána do velké haly (dozvukové komory). V této hale byl na jedné stěně reproduktor a na protilehlé stěně další mikrofon, který nahrával zvuk s ozvěnou z haly. Takto vzniklý signál pak smíchali s původním signálem (zdroj [28]).

Dnes naštěstí díky výkonu moderních technologií můžeme výše popsaného zvukového efektu ozvěny dosáhnout pomocí malé hardwarové krabičky (efektu) přidané za mikrofon. V případě použití softwarových zvukových pracovních stanic přidáním VST zvukového efektového zásuvného modulu do efektového racku pro daný kanál.

Většina zvukových nadšenců dnes využívá v domácích podmínkách k tvorbě hudby softwarových zvukových pracovních stanic. V těchto případech je mnohem pohodlnější použití levných softwarových efektů místo drahých hardwarových řešení. Lze je dokonce aplikovat při nahrávaní v reálném čase - libovolně vypínat, zapínat, modifikovat či automatizovat jejich parametry. Limitaci jsou pouze výpočetní nároky jednotlivých softwarových efektů.

Domácí produkce se již může zcela obejít bez hardwarových efektů. Výsledky takto vyrobených skladeb se stále více přibližují studiovým nahrávkám upraveným drahými hardwarovými přístroji v profesionálních hudebních studiích. Některé nedostatky softwarových efektů však zůstávají nepřekonány. Například u efektů jako jsou kompresory, předzesilovače,

limitery nebo zkreslovací pedály je rozdíl ve výsledném zvuku pořád dosti znatelný. Na druhé straně stojí cenová dostupnost, jednoduchost instalace, "skladnost" a mnohdy širší funkcionalita softwarových řešení. Proto čím dále více amatérských hudebníků dává přednost právě jim.

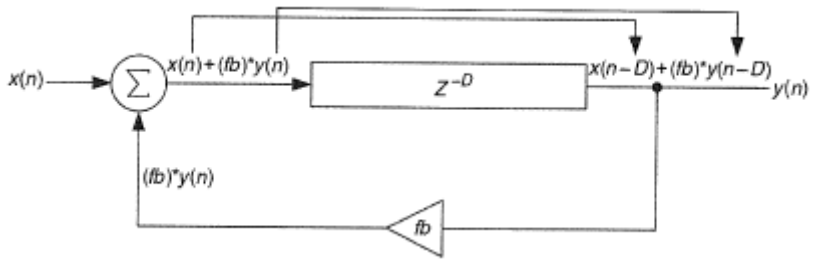
Druhů a způsobů použití zvukových efektů je celá řada a jejich popis by vydal na celou knihu. Níže stručně popisuji alespoň ty nejznámější. To by mělo sloužit čtenáři jako výchozí bod pro následné hlubší objevování světa zvukových efektů.

2.2.1 **Delay efekty**

Někdy nazývaný také echo nebo ozvěna je jeden z nejzákladnějších audio efektů a zároveň jeden z nejdůležitějších. Jeho princip se používá v implementaci mnoha dalších typů efektů, jako jsou například reverb a chorus. Jádrem delay efektu je zpožďovací člen, který zaznamenává vstupní signál a s nějakým zpožděním ho reprodukuje na výstupu. Přidáním parametru zpětné vazby (feedback) se část výstupního signálu přivádí zpátky na vstup, čímž je dosaženo několikanásobného opakování původního signálu se sestupnou tendencí hlasitosti. V podstatě se jedná o imitaci dobře známého zvukového jevu ozvěny.

Již v 50.tých letech minulého století se tento efekt začal hojně používat. V té době byl delay sestrojen za pomocí několika pásek. Příchozí signál nahrávali pomocí záznamové hlavy na nekonečný záznamový pásek. V přístroji bylo několik čtecích hlav, které postupně záznam četly a mixovaly s původním signálem. Tím dosáhli opakování původního signálu. Takto fungující přístroje se nazývaly pásková analogová echa. V 70.tých letech přišla éra digitálního zpracování signálů, která přinesla digitální verzi tohoto efektu. Digitální verze nabízí mnohem širší škálu funkcionality, zejména možnost použití mnohem delších intervalů ozvěn. Digitální forma umožnila jednodušší přechod na PC, kde může být delay efekt implementován jako zásuvný zvukový modul.

Základní princip digitálního delay efektu je zobrazen na obrázku 2.3. Když je feedback parametr nastaven na nulu, efekt poskytne pouze jeden zpožděný signál. V momentě aplikace feedbacku se však část signálu vrací před zpožďovací člen a jeho opětovným zpracováním umožňuje vícenásobný delay efekt. Takto fungující delay efekt popisuje rovnice 2.1. Kde x(n) je původní signál, fb parametr pro nastavení počtu opakování a D hodnota intervalu ozvěny.



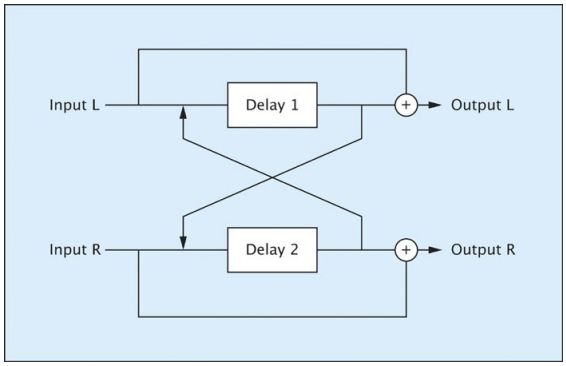
Obrázek 2.3: *Základní princip delay efektu*

$$y(n) = x(n - D) + fb * y(n - D) \quad (2.1)$$

Typické parametry delay efektů:

- Time delay - Interval ozvěny (za jak dlouho zazní duplikovaný signál),
- Feedback - Počet opakování (v závislosti na Time delay)
- Panning - Stereo rozložení ozvěny
- Tempo Sync - Synchronizace delay efektu s templem skladby. Time delay se například automaticky nastaví na každou osminovou dobu. Zpožděný signál pak kopíruje tempo skladby, což zní mnohdy lépe.
- Dry/Wet - poměr originálního signálu a zpožděného

Krom základního použití pro dosažení ozvěny existuje několik dalších. Prvním z nich je PinPong delay efekt. Schéma fungování je na obrázku 2.4. Používá dvě mono delay jednotky pro každý kanál jeden. Vstupní signál je první jednotkou zpracován a její výstup pak „krmí“ jednotku druhého kanálu. Tím je dosaženo „Ping pong“ stereo efektu.



Obrázek 2.4: *Princip PingPong delay efektu*

Dalším použití delay efektu slouží pro "roztáhnutí" mono zvuků do sterea. Můžeme nastavit původnímu mono signálu jiné parametry delay pro každý kanál. Tím docílíme, že signál pro levý kanál bude trochu jiný než signál pro pravý. Posluchač bude vnímat takový zvuk jako stereo.

Také známý Flanger efekt je prakticky Delay fungující s velmi krátkým intervalem zpoždění (typicky 1-10 ms). Lidské ucho tak krátký interval nerozeznává jako dva rozdílné zvuky a výsledkem je specificky znějící zvuk. Používá se hojně pro kytaru nebo bicí. Flanger je někdy mylně přirovnáván k dalšímu efektu Phaseru (Phase shifter). Ten sice také pracuje na principu zpoždění signálu, ale ne pomocí aplikace delay efektu. Konstruuje se pomocí několika all-pass filtrů. Z pohledu uživatele jsou výsledné zvukové efekty podobné, ale z technické stránky jde zcela o jiný přístroj.

Na obrázku 2.5 jsou tři ukázky delay efektů. Zleva staré páskové echo od firmy Roland Space Echo RE-201, uprostřed digitální delay JOYO JF-08 a vpravo softwarový VST delay plugin SonicXTC.

Obrázek 2.5: *Zleva Páskové Ech RE-201, digitální JOYO JF-08 a VST zásuvný modul EchoFlux*

2.2.2 **Reverb efekty**

Reverb je zvukový efekt simulující ozvěnu v místnosti. Proto se také někdy nazývá hall efekt. Pří šíření zvukových vln v uzavřených prostorách dochází po setkání s překážkou k částečnému odrazu zvuku. Prvním faktorem ovlivňujícím charakter odraženého zvuku je velikost místnosti. K tomu nu reverb efektu slouží parametr Reverb-Time, který simuluje čas, který musí zvuková vlna urazit od zdroje k překážce a zpět. Další parametr Frequency cut-off slouží k simulaci různých odrazových vlastnosti materiálu překážky. Například dřevo odráží jiné frekvence než kovy, tedy různé hodnoty Frequency cut-off parametru simulují různé odrazové materiály. Dozvuk je přirozenou součástí akustického prostoru, proto zvuky bez odrazu, jako jsou třeba zvuky syntetizérů, by bez aplikace reverbu zněly lidskému uchu velmi nepřirozeně. Reverb se také používá všude tam, kde chceme docílit zdání větší prostorovosti. Velice často je aplikován na vokály, strunné, dechové či perkusní zvuky. Výjimkami jsou basové stopy a "kopák"(velký basový buben). Pokročilé a zejména pak softwarové Reverby často poskytují předvolby svých parametrů simulující dané prostředí. Typickými předvolbami jsou hall (hala), stage (stadion) nebo cathedral (katedrála). To umožňuje uživateli dosáhnout požadovaného efektu bez zdlouhavého nastavování jednotlivých parametrů. Za zmínku stojí také velice častý parametr Predelay, který nastavuje zpoždění upraveného signálu s ozvěnou oproti původnímu signálu. Pomocí správného nastavení parametru predelay a poměru obou signálů pomocí dalšího parametru Dry/Wet se dá dosáhnout velmi přirozeně znějícího efektu.

Typické parametry reverbů:

- Reverb Time - Čas za který se ozve opakovaný signál
- Predelay - Slouží pro nastavení odstupu upraveného signálu od původního
- Dry/Wet - Nastavení poměru původního a upraveného signálu
- Frequency cut-off - Regulace frekvencí obsažených v odraženém zvuku

Pro jednotlivé parametry existuje mnoho různých označení, proto mnohdy vypadá, že reverb umí něco nového, avšak po bližším prozkoumání se vlastně jedná pouze o jiné pojmenování jedné ze základních funkcí uvedených výše. To ostatně platí napříč všemi zvukovými efekty. Na obrázku 2.6 je vlevo hardwarový reverb efekt ve formě kytarového pedálu HT-Reverb Pedal od firmy Blackstar a vlevo pak pokročilý softwarový VST reverb IQ-Reverb od firmy HOFA. Ačkoli produkty slouží zcela k jiným účelům, z hlediska zpracování zvuku poskytují velice podobné funkce. Navíc softwarová verze disponuje vynikajícím

grafickým rozhraním s velkým množstvím nastavení. Pro představu je dnes hardwarová verze přibližně dvakrát dražší než ta softwarová.

Obrázek 2.6: *Vlevo hardwarový HT-Reverb Pedal a vpravo VST reverb IQ-REverb*

2.2.3 Chorus efekty

Chorus efekt vzniká, když dva nebo více zvuků podobné barvy a výšky (ale nikdy úplně stejné) zní současně. To nastává například v pěveckých sborech, kde zpívá mnoho zpěváku najednou (odtud je odvozen název efektu).

Ve zvukové technice chorus efekt značí modulační efekt, který původní signál znásobí (dvakrát nebo více) a kopie mírně zpozdí a rozladí. Velmi často se integruje do digitálních pian, kytarových zesilovačů. Velmi známé jsou také chorus kytarové pedály. Dodávají zvuku hloubku a prostorovost. Efekt chorus může být simulován analogovými nebo digitálními procesory. Často jsou také simulovány zvukovými moduly na počítačích.

Typické parametry:

- Depth – udává hloubku efektu, množství aplikované modulace výšky tónu
- Rate – je frekvence modulace nejčastěji v rozmezí 0,1 – 10 Hz
- Feedback – posílá upravený signál zpátky na vstup a tím zvýšit efekt chorusu
- Dry/Wet – Poměr původního signálu a upraveného (někdy také Stage)
- Predelay – Jak daleko v čase bude zvuk chorusu od originálu

Zástupce této kategorie efektů můžete vidět na obrázku

Obrázek 2.7: *Vlevo softwarový Chorus z Cubase 5 a vpravo kytarový pedál Fender Chorus Pedal*

2.2.4 **Fitry**

Zvukové filtry jsou prakticky na frekvenci závislé zesilovače. Většinou pracují ve frekvenčním pásmu 0 - 20kHz. Základní podstata filtrů je zesílit, propustit nebo zeslabit (negativní zesílení) určité frekvenční pásmo zvukového signálu. Typickými zástupci jsou low-pass filtry, které disponují paramatrem cut-off frequency. Tedy low-pass filter propustí všechny frekvence pod cut-off frekvencí a frekvence nad touto hranicí utlumí. Opačně funguje hi-pass filtr, který propouští frekvence nad cut-off hranicí a utlumuje ty pod ní. Další dvojící filtrů jsou band-pass a band-reject filtry. Ty disponují dvěmi cut-off frekvencemi. Band-pass propouští frekvence mezi nimi a utlumuje zbytek. Band-reject filtry dělají přesný opak, tedy utlumují pásmo mezi cut-off frekvencemi a zbytek signálu nechávají projít.

Zvláštním typem filtrů jsou all-pass filtry, které propouští všechny frekvence, ale ovlivňují fázi jednotlivých sinusových složek signálu. To se například využívá při implementaci Phaser efektu. Toto je výčet jen těch nejzákladnějších typů filtrů. Vybral jsem ty nejčastěji užívané při implementaci VST modulů. Některé další typy také zmiňuji u popisu ekvalizérů, které jsou prakticky na filtrech postaveny. Vice se o filtrech můžete dočíst zde [11] .

Filtry většinou disponují parametrem pro ovládání resonance. Ta byla nechtěným prvkem u filtrů ve starých hardwarových syntetizérech. Jedná se o úzké frekvenční pásmo blízko cut-off frekvence, které je zesíleno. V dnešních přístrojích může být tento jev jednoduše odstraněn. Avšak způsobuje speciální zvukový efekt. Pohyb cut-off frekvence společně s vysoko nastavenou resonancí způsobuje efekt podobný Phaseru. Proto se i u dnešních filtrů stále implementuje.

Posledním velice častým parametrem je strmost filtru. Ideální filtr by se choval tak, že například u low-pass filtrů by všechny frekvence pod cut-off frekvencí byly propuštěny s nulovým zeslabením a ty nad ní zcela utlumeny. To není bohužel konstrukčně možné. Proto se zavadí strmost, která udává kvalitu filtru. Tedy čím více se filtr blíží ideálnímu případu, tím větší strmost poskytuje. U softwarových filtrů je limitována výkonem procesoru. Typickými hodnotami jsou 6dB/octave, 12dB/octave a 24dB/octave. Cifra značí, o kolik decibelů za jednu oktávu nad nebo pod cut-off frekvencí je signál zeslaben. Teorii a obrázky demonstrující problematiku strmosti filtrů můžete najít zde [12].

Typické parametry filtrů:

- Cut-off frequency - frekcence pod nebo nad kterou filtr začíná utlumovat nebo zesilovat
- Resonance - množství obsažené resonance
- Strmost - strmost filtrů

Existují i samostatné hardwarové filtry, nicméně nejvíce se využívají jako moduly do větších zařízení. Příkladem mohou být filtrovací jednotky v syntetizérech. Jsou také nepostradatelnou součástí ekvalizérů. Stejně tomu je na platformě počítačů. Na obrázku 2.8 můžete vidět hardwarový low-pass filtr efekt MF-101 Lowpass Filter od firmy Moog a VST zásuvný modul firmy Steinberg obsažený v pracovní stanici Cubase 5.

Obrázek 2.8: *Vlevo MF-101 a vpravo VST filtr Tonic*

2.2.5 **Ekvalizéry (EQ)**

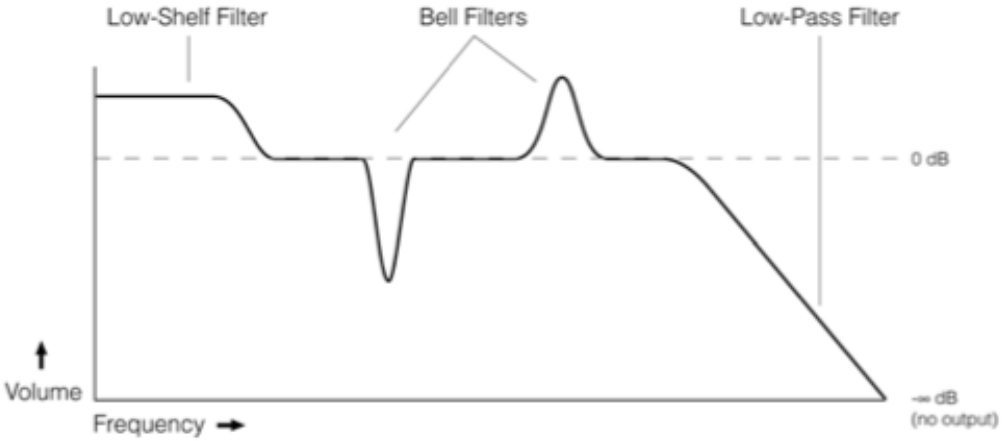
Ekvalizér je zařízení sloužící k úpravě frekvenční charakteristiky zvukového záznamu. Bývá často základní výbavou všelijakých přístrojů vydávajících zvuk. Najdete je od televize přes zvukovou kartu počítače po mp3 přehrávače.

Ekvalizér využívá nejčastěji jeden nebo více jednoduchých lineárních filtrů pro rozdělení frekvenčního spektra do pásem, kterým lze pak nastavovat různé úrovně hlasitosti. Tímto můžeme například vhodně upravit zvuk nahrávky v závislosti na momentálním reprodukčním systému. Ve své nejjednodušší verzi se ekvalizéry nachází na téměř každém Hi-fi systému, kde umožňují nastavení poměru basových, středových a výškových frekvencí. Pokročilejšími typy jsou například ekvalizéry parametrické, polo parametrické a grafické.

Základní ekvalizační technika využívá high-pass a low-pass filtry s velkou strmostí. Přikladem může být multifunkční přístroj Focusrite ISA430, zmíněný v článku detailně popisujícím roli filtrů v ekvalizaci [13]. Jeho ekvalizér disponuje filtry se strmostí 18dB/octave. Ty postupně redukují frekvence pod nebo nad uživatelem nastaveným cut-off parametrem, zatímco zbylé frekvence nechávají téměř nepozměněny. To se hodí k utlumení nejnižších a nejvyšších frekvencí. Jedná se o takzvanou subtraktivní ekvalizaci, kdy se určité frekvence utlumují. Opakem subtraktivní ekvalizace je aditivní, kdy některé frekvence naopak zesilujeme. Pro tyto účely se zmíněné low-pass a high-pass filtry nehodí. Kdybyste totiž obrátili jejich logiku a místo utlumení frekvencí je začali zesilovat, zesilovaly by se rychlostí vycházející ze strmosti filtru. Pro Focusrite ISA430 by to bylo -18dB/octave. To by velice rychle vedlo ke zkreslení signálu.

Pro tvarování dolních a horních frekvencí spektra, proto potřebujeme jiný typ filtrů. Tím jsou shelving filtry. Ty jsou navrženy tak, aby upravily hlasitost všech frekvencí za uživatelem nastavenou shelving frekvencí (obdoba cut-off frekvence) na určitou hodnotu. Počáteční chování filtru je téměř stejné jako u hi-pass a low-pass filtrů. Avšak navíc zde figuruje krom shelving frekvence také stop frekvence, na které se trend utlumení nebo zesílení zastaví a hlasitost se ustálí. V přístrojích je tato stop frekvence často interpretovaná jako Gain, tedy o kolik decibelů budou dané frekvence zesíleny nebo zeslabeny.

Posledním druhem filtrů užívaných při ekvalizaci jsou peak filtry (někdy také nazývané bell filtry). Ty se hodí, když je třeba upravit nějakou frekvenci uprostřed spektra. Díky nim je možné zaměřit konkrétní frekvenci a pomocí parametrů Gain a Q přesně nastavit úpravu signálu v konkrétním pásmu. Gain udává o kolik decibelů bude dané pásmo utlumeno nebo zesíleno a Q parametr (Quality parameter) nastavuje již zmíněnou strmost filtru S rostoucí Q se "špička" zužuje, tedy zvyšuje se kvalita filtru. Obrázek 2.9 ukazuje, jak jednotlivé druhy filtrů působí na frekvenční spektrum zvuku. Více informaci o peak filtrech naleznete zde [14].



Obrázek 2.9: *Znázornění efektu různých typů filtrů používaných při ekvalizaci [14]*

Prvním typem EQ jsou grafické ekvalizéry. Vstupní signál je rozdělen pomocí sady filtrů do několika frekvenčních pásem. Přístroj poskytuje pro každý band potenciometr, který slouží k nastavení hlasitosti jednotlivých části frekvenčního spektra signálu. Příklad grafického ekvalizéru je na obrázku 2.10. Grafické se jmenují proto, že rozdělení do mnoha pásem a přechod od rotujících potenciometrů k vertikálním umožňuje jejich uspořádání tak, aby připomínaly grafické zobrazení frekvenčního spektra signálu.

Obrázek 2.10: *Vlevo hardwarový grafický ekvalizér, vlevo VST EQ modul*

Druhým velice oblíbeným typem ekvalizérů jsou parametrické ekvalizéry. Plně parametrický ekvalizér obsahuje pro každé pásmo právě tři parametry Frequency, Q (strmost) a Gain. To poskytuje uživatelům nástroj pro upravování různých frekvenčních částí signálu s téměř chirurgickou přesností. Více informací najdete zde [14]. Na obrázku 2.11 je hardwarový hi-endový parametrický ekvalizér Ashly PQX-572 a na obrázku 2.12 jeho

softwarová obdoba VST zásuvný modul AC-Q firmy Audiocation, který je dokonce dostupný zdarma.

Obrázek 2.11: *Hardwarový parametrický EQ Ashly PQX-572*

Obrázek 2.12: *VST parametrický kompresor AC-Q*

2.2.6 **Kompresory**

Komprese dynamiky zvukového signálu je proces používaný ve zvukovém inženýrství ke zmenšení dynamického rozsahu zpracovávaného signálu. Pro tyto účely slouží zařízení zvané kompresor. Kompresory mohou být buď hardwarové, ty se přidávají jako modul do studiových efektových racků, nebo softwarové, které se nahrávají do počítačových zvukových stanic DAW. Příklady obou typů můžete vidět na obrázku 2.13.

Obrázek 2.13: *Vlevo hardwarový kompresor PuingCHild vlevo VST Rough Rider*

Kompresor zmenšuje rozdíly mezi hlasitými a tichými pasážemi nahrávky. Používá se kvůli omezenému dynamickému rozsahu záznamových médií. V momentě kdy nahráváme mnoho různých kanálů (např. symfonický orchestr), méně hlasité nástroje by se mohly ztratit v hluku ostatních. Proto je třeba dynamiku jednotlivých nástrojů trochu zkomprimovat, což umožní jejich vměstnání do výsledného mixu. Ve studiích se kompresory používají jak na jednotlivé nástroje, tak na výsledný mix. Často to usnadňuje mixáž a přidává výsledné skladbě

celistvost. Například basová kytara a bicí mají velký dynamický rozsah (někdy i přes 100dB), proto se zařazením kompresoru do signálové cesty docílí lepší průraznosti a čitelnosti po smíchání s ostatními nástroji. Pomocí vhodné aplikace se dají napravit dynamické chyby vzniklé „nekázní“ hráče nebo zpěváka. Často se také komprimuje lidský hlas, který obsahuje velmi hlasité, ale také i velmi tiché složky. Ten pak vynikne oproti okolnímu hluku či podkresové hudbě.

Kompresor dynamiky upravuje zvukový signál změnou výstupní hlasitosti závisle na vstupní. Pokud vstupní hlasitost překročí nastavený práh (Threshold), sníží se úroveň výstupního signálu v nastaveném poměru (Ratio). Přičemž Rychlost nástupu a ústupu komprese je řízená nastavením parametrů Attack a Release.

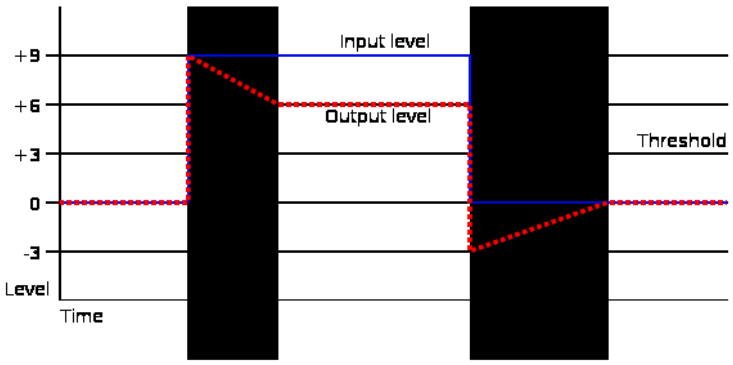
Častým parametrem dostupným uživateli bývá také Knee. Jedná se pokročilejší funkci kompresorů, která umožňuje přepínat mezi „tvrdým“ (Hard knee) a „měkkým“ (Soft knee) průběhem komprese. Při nastaveném měkkém průběhu kompresor nezačne po dosažení Threshold komprimovat v nastaveném Ratio ostrým skokem, ale pozvolným náběhem funkce s minimálním poměrem. Nastaveného poměru dosáhne až v jisté úrovni nad Threshold. Zvuk pak působí často přirozeněji, protože rozdíl mezi komprimovanou a nekomprimovanou části projevu není tak patrný.

Typické parametry:

- Threshold - Hranice hlasitosti, která slouží k aktivaci komprese
- Attack - Rychlost zapnutí komprese po překročení Threshold (obvykle v milisekundách)
- Release - Rychlost vypnutí komrese po poklesu signálu pod Threshold (obvykle v milisekundách)
- Input - Zesílení signálu před kompresí
- Output - (mnohdy také Make-up) kompenzace hlasitosti signálu po kompresi
- Knee - (mnohdy také Soft-Knee) přepnutí agresivního nástupu komprese Hard-Knee na plynulejší Soft-Knee

Příklad:

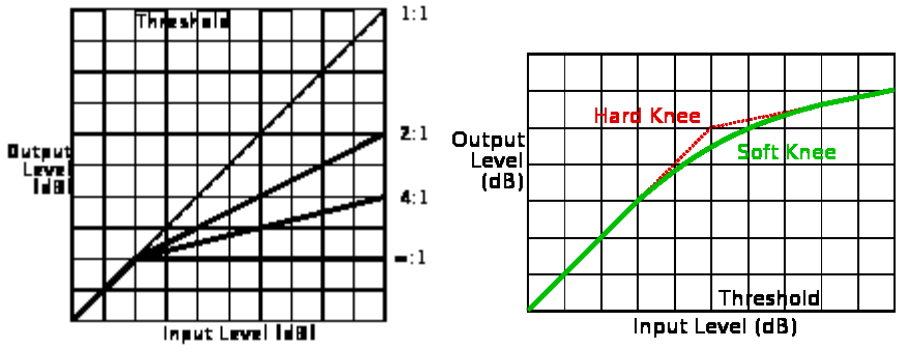
Kompresor je nastaven na Threshold na 3 dB a Ratio 2:1, tedy vše, co přijde na kompresor hlasitější než 3 dB, bude zkomprimováno. Situaci můžete vidět na obrázku 2.14.



Obrázek 2.14: *Obrázek demonstrující průběh komprese [15]*

Modrý input signál překračuje Threshold o 6 dB, poměr Ratio máme nastaven na 2:1, tedy se úroveň hlasitosti části signálu překračující Threshold sníží právě dvakrát. 6/2 = 3dB, což znamená, že input signál klesá po dobu nastavenou parametrem Attack na 6dB. V momentě kdy úroveň vstupního signálu vrátí pod hodnotu Threshold, nastává fáze Release, která opět navýší hlasitost signálu do nuly (za dobu nastavenou Release parametrem).

Funkce zvukových kompresorů se často znázorňuje grafem závislosti hlasitosti výstupního signálu na hlasitosti vstupního signálu. Tento graf můžete vidět na obrázku 2.15 vlevo. Vpravo pak graf demonstrující funkci Soft-Knee.



Obrázek 2.15: *Vlevo funkce kompresoru a vpravo demonstrace funkce Soft Knee[15]*

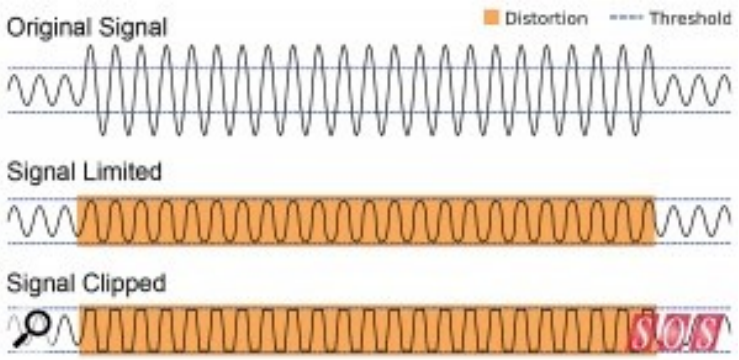
Kompresor nemusí nutně upravovat celé frekvenční spektrum signálu, ale třeba jen některé jeho části. Takto pracující zařízení se nazývá vícepásmový kompresor (Multiband compressor). Při aplikaci velké komprese dynamiky se můžou vyskytnout nežádoucí jevy, jako je například „pumping“. Ten se může objevit v části skladby, kde je dlouhý tón zpěváka a pod ním silná rytmická basová linka. Po aplikaci celkové komprese se sice sníží dynamika basových úderů, ale zároveň je komprimován zpěvákův tón. Výsledkem bude zeslabení zpěvákova tónu při basových úderech, kdežto mezi nimi nikoliv. Vícepásmový kompresor tento jev šikovně řeší rozdělením signálu do několika pásem, nad kterými pak aplikuje různé úrovně komprese. Tedy vysoká úroveň komprese basového pásma neovlivní pásmo zpěváka (pokud zpěvák nezpívá Bas).

Existuje celá řada použití kompresorů v audiotechnice. Jako jsou speciálně navržené kompresory k odstraňování sykavek takzvané De-essery, paralelní kompresory pro paralelní kompresi, expandery atd. Pochopitelně všechny najdete jak ve formě hardwarového přístroje i ve formě softwarové. Mnohdy jde také technicky o kompresory, jen s jiným marketingovým označením, jako je tomu třeba u maximizérů. Já zde popíši jen dva základní druhy použití kompresorů. Více informací o kompresorech naleznete zde [15] nebo zde [17].

Základní využití kompresorů je pro „lehčí“ kompresi cca do Ratio 4:1. Slouží pro ohlídání dynamiky nástrojů a vyvážení zpěvu. „Výrazná“ komprese někdy i Ratio nad 10:1 se používá pro záměrné ovlivnění či zabarvení zvuku nástroje. Toto se například aplikuje na baskytaru, která je poté v nahrávce výraznější s menšími výkyvy v hlasitosti. V posledních letech se začala komprese ve velké míře aplikovat ne jen na jednotlivé zvukové stopy, ale také při masteringu (finální úpravě celé skladby). Některé dnešní skladby mají až směšný dynamický rozsah, tedy rozdíl mezi nejtišší a nejhlasitější části skladby (mnohdy jen pár decibelů). Tyto skladby sice zaujmou svou plností, ale po chvíli se ucho z nedostatku dynamiky začne nudit. Okolo této problematiky vznikla v hudebních kruzích velice známá „Loudness war“. Kde proti sobě stojí poptávka po stále větší aplikací komprese pro masivnější nahrávky a zkreslení původního zvuku či dokonce jeho poškození velkou kompresí. Více o tomto zajímavém tématu ze světa hudebního průmyslu se můžete dočíst zde [16].

Dalším velice známým využitím kompresorů je limiter. Limiter je kompresor s Ratiem na maximum (teoreticky nekonečno) a Attack a Release na minimum (okamžitá reakce). Z hlediska procesu odehrávajícím se na pozadí je limiter totožný s kompresorem, jen díky vysokému poměru komprese a hodnot Attack a Release, prakticky nepustí nic nad hranici threshold. Pro pochopení, proč je toto chování vyžadováno, je třeba znát další velice známý pojem obzvláště ve světě digitálního zpracování signálu - clipping. K tomu dochází, když hlasitost zvuku přesáhne možnosti svého média (typicky je na monitorovacích prvcích hlasitosti vyobrazena červenou barvou). V momentě kdy již stroj nemá prostředky k zaznamenání takového signálu, vše co tuto hranici překročí, nahradí svou maximální možnou hodnotou. Takto ořezaný signál můžete vidět na obrázku 2.16. Očividně zde dochází ke ztrátě informace a zkreslení. Na stejném obrázku můžete vidět šetrnost techniky limitování signálu. To sice upraví dynamiku původního signálu, nicméně jeho tvar zůstává podobný.

Clipping nemusí být vždy negativní jev, v moderní hudbě se často využívá k ještě většímu zesílení zvuku i za cenu ztráty jakési části informace. Můžete narazit například na VST distortion efektové moduly nebo clippery.



Obrázek 2.16: *Rozdíl mezi clippingem a aplikací limitéru [17]*

2.2.7 **Distortion efekty**

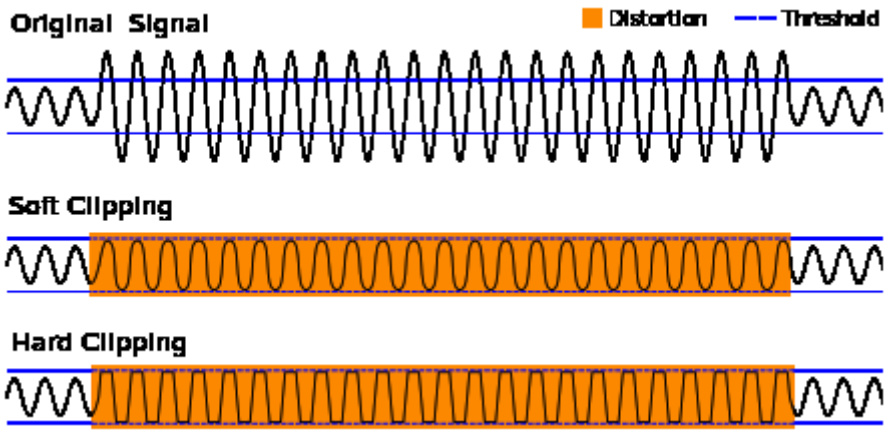
Ve spojitosti se zpracováním signálu se dá distorze přeložit do češtiny jako zkreslení, které obecně znamená nechtěnou změnu vlastností signálu. Pochopitelně se mu zvukoví inženýři v mnoha oblastech zpracování zvuku snaží vyhnout. To ale neplatí v kontextu hudební produkce. Zde se naopak vyrábí spousta distorzních zařízení, které umožňují pomocí distorze vytvářet „teplejší“, „špinavější“ a „rozmazanější“ zvuky. Jedná se o velkou míru komprese špiček původních signálů, která má za následek vytváření vyšších harmonických frekvencí (někdy také nazývané alikvoty nebo alikvotní tóny). Nejznámější je asi nelineární distorze vnikající v zesilovačích. Nelineární znamená, že výsledný signál není přímo úměrný původnímu. Jsou totiž generované nové frekvence, které původně neobsahoval.

Distorze je například typická pro zvuk elektrické kytary v rock and rollu. Také ji najdete v mnoha moderních tanečních hudebních žánrech, jako jsou neurofunk nebo electro house.

Rozdělení distortion efektů vychází z použité techniky pro dosažení zkreslení a výsledného zvuku. Ačkoli je mnohdy distortion označení pro všechny tyto efekty, je také jednou jejich skupinou.

Overdrive (přebuzení) distorze bývá dosaženo kompresí zvukového signálu v lampových zesilovačích nebo elektronkách. Množství a zvuk takto dosažené distorze je závislý na hlasitosti. Při vyšších hlasitostech je distorze „špinavější“ než při nižších. Zatímco Overdrive efekty produkují soft-clippingu, druhý typ - Distortion efekty ořezávají špičky signálu kompletně - hard-clipping, to je typické když zesilovače překročí své limity. Nejmohutnější distorzi poskytují Fuzz efekty („fuzzboxy“). Všechny tři druhy distorze bývají simulovány hardwarovými i softwarovými efekty.

Tato problematika je velice obsáhlá a pro správné odlišení jednotlivých typů je třeba znát mnoho technických detailů. Pro uživatelé je podstatný výsledný zvuk, který je u každého typu znatelně odlišný. Pro programátora, který chce distortion efekt nasimulovat v prostředí počítače, jsou stěžejní pojmy soft a hard-clipping (obrázek 2.17) na základě kterých, se dá odvodit, jaký algoritmus by vyžadovanému chování nejlépe vyhovoval. Pro více informací o distortion efektech čtěte zde [24].



Obrázek 2.17: *Rozdíl mezi Soft-clipping a Hard-clipping*

Na obrázku 2.17 je soft-clipping, který signál komprimuje a jeho zkreslení není tak velké. Pod ním vidíte hard-clipping, který zkosí všechny špičky signálu. (Podobnost s obrázkem 2.16 není náhodná).

2.3 Možnosti implementace VST zásuvných modulů

Vhodná volba implementačních nástrojů je dalším stěžejním aspektem každého softwarového projektu. Samozřejmě i pro vývoj zvukových zásuvných modulů existuje více možností. Níže popisuji a komentuji ty nejznámější z nich.

Základem celé technologie je framework VST SDK vydaný firmou Steinberg. Nabízí se možnost použít právě ten. VST SDK je napsán v nespravovaném C++. Nespravovaný C++ má bezesporu mnoho pozitivních stránek, jako jsou rychlost, kompilace programu rovnou do strojového kódu, nezávislost na platformě a stále velká komunitní podpora. Pro mladší programátory zvyklé na vlastnosti jazyků, jako jsou například velmi oblíbená Java a C#, se může C++ zdát přespříliš komplikovaný, s velkou časovou náročností vývoje. Roli také hraje prostředí, ve kterém je programátor zvyklý pracovat a nástroje, které mu usnadňují život a zrychlují razantně jeho práci (například ReSharper ve Visual Studiu). Problematickým místem nespravovaného C++ je vývoj grafického rozhraní. A i když Steinberg pro tyto účely vydal velice dobrou knihovnu VSTGUI, se kterou se dají naprogramovat opravdu impozantní uživatelská rozhraní, nemůže se rovnat s dnešními technologiemi pro vytváření grafických rozhraní, jako jsou WinForms nebo velice moderní Windows Pressentation Foundation (WPF).

Protože má tato práce sloužit převážně „mladým“ programátorům, rozhodl jsem se porozhlédnout po jiných možnostech implementace mého VST modulu. Jak co nejefektivněji investovat čas strávený vývojem a hlavně jak zpřístupnit programátorům moderní vývojářské technologie. Seznam možností implementace není velký, proto uvádím stručně každou z nich.

2.3.1 Steinberg VST plugin SDK

Virtual studio technology plugin Software Developement Kit je esenciální prvek při tvorbě VST zásuvného modulu. Jedná se o skupinu nespravovaných C++ tříd poskytující základní komunikaci a struktury vývojářům třetích stran pro vývoj jejich vlastních VST modulů. I když uvádím další frameworky, je třeba si uvědomit, že i ty musí vyhovovat základnímu konceptu VST technologie a obsahovat alespoň nejzákladnější třídy a rozhraní z VST SDK, na základě kterých hostovací aplikace zahájí komunikaci se svým modulem.

Zdrojový kód VST SDK je nezávislý na operačním systému. Avšak při vývoji vlastního VST modulu je nutno použít náležitou verzi finálního produktu. Je to proto, že zvukové aplikace v různých operačních systémech vyžadují různé formy VST modulů. Například VST zásuvný modul pro Windows je více vláknová DLL knihovna, kdežto pro MAC OS X je to archív. VST SDK balík lze získat zdarma na webu firmy Steinberg. Po bezplatné registraci je uživateli umožněn vstup do sekce vývojářů třetích stran. Zde je možno stáhnout 4 různé balíky.

VST Audio Plug.Ins SDK

Tento balík můžete stáhnout ve dvou verzích 2.4 a 3.5. Převládají stále VST moduly napsané pod verzi 2.4. a zvukové pracovní stanice jej podporují nejlépe. Avšak během psaní mé práce Steinberg oficiálně ukončil distribuci verze 2.4. Původně jsem zde psal, že doporučuji začít právě s ní (kvůli dostupnosti informací) a až po pochopení základních procesů, přejít na verzi vyšší. To se však díky tomuto tahu Steinbergu mění a je třeba začít vyvíjet pod verzi novější. Bohužel do dnes jsem k nové verzi nenašel žádné návody, blogy či demonstrace vývoje. Programátor je tedy odkázán pouze na původní dokumentaci.

ASIO SDK

Je Framework a popis technologie Audio Stream Input/Out vydané společností Steinberg. ASIO program je ovladač zvukové karty, který umožňuje přímý přístup k hardwarů a odbourává problémy s latencí, způsobené několika vrstvami zpracování zvuku na platformě Windows.

VST Module Architecture

Je objektově orientovaný model rozhraní nezávislý na platformě a kompiléru. Specifikuje, jak komponenty musí vypadat a jak jsou vytvářeny hostovací aplikací. Jedná se o obdobu VST zásuvných modulů. Nicméně podpora ze strany hostovacích aplikací je podstatně nižší.

2.3.2 JUCE (Jules‘ UtulityClass Extensions)

Tento projekt komerčně vypustil Julian Storer v roce 2005. Je to další C++ Framework rozšiřující možnosti původního VST SDK. Podporuje platformy Mac OS X, Windows, Linux, iOS a Android. Zejména podpora operačního systému Android je zde lákavá. Příjemná je také GNU Public Licence, která umožňuje kód použít v open-source projektech zcela zdarma.

V momentě kdy chcete kód použít pro komerční closed-source programy, je třeba zakoupit komerční licenci. Díky modularitě frameworku je opravdu jednoduché využití jeho tříd uvnitř vašeho projektu. JUCE obsahuje přes půl milionu řádků kódu a přes sto tříd. Podporu poskytuje web projektu [29], který je docela přehledný a obsahuje mnoho užitečných informací.

2.3.3 jVSTwRapper

Obaluje základní třídy frameworku VST SDK tak, aby bylo možno psát VST zvukové zásuvné moduly v programovacím jazyce JAVA. Kromě psaní zásuvných modulů ve standardu VST umožňuje výrobu pod standardem Audio Unit (AU) a LADSPA(Linux Audio Developer's Simple Plugin API). Obdobně jako VST je AU další standard pro zvukové moduly běžící na Mac OS X. LADSPA by mohl zajímat zaryté milovníky Linuxu, ačkoli jsem nikdy neslyšel o nějaké seriózní zvukové pracovní stanici běžící pod Linuxem.

Princip jVSTwRapper je jednoduchý. V podstatě se zkompiluje část původního kódu frameworku VST SDK, který přemostí nativní volání hostující aplikace. jVSTwRapper pak vystaví rozhraní k implementaci potřebných tříd pro zpracování zvukového signálu a grafického rozhraní v jazyce JAVA. Díky nezávislosti Java Runtime Machine na platformě, lze takto napsané VST zásuvné moduly, bez další práce, jako je rekompilace, psaní specifického grafického rozhraní pro danou platformu atd., spouštět na mac/win/linux platformách. Pro milovníky Java jazyka je toto jediná (kterou jsem objevil), ale za to velice šikovná cesta k vývoji na platformě nezávislých VST zvukových modulů Informace jsem čerpal z webu projektu http://jvstwrapper.sourceforge.net/.

2.3.4 IPlug C++ code framework for VST and AU audio plugins and GUIs

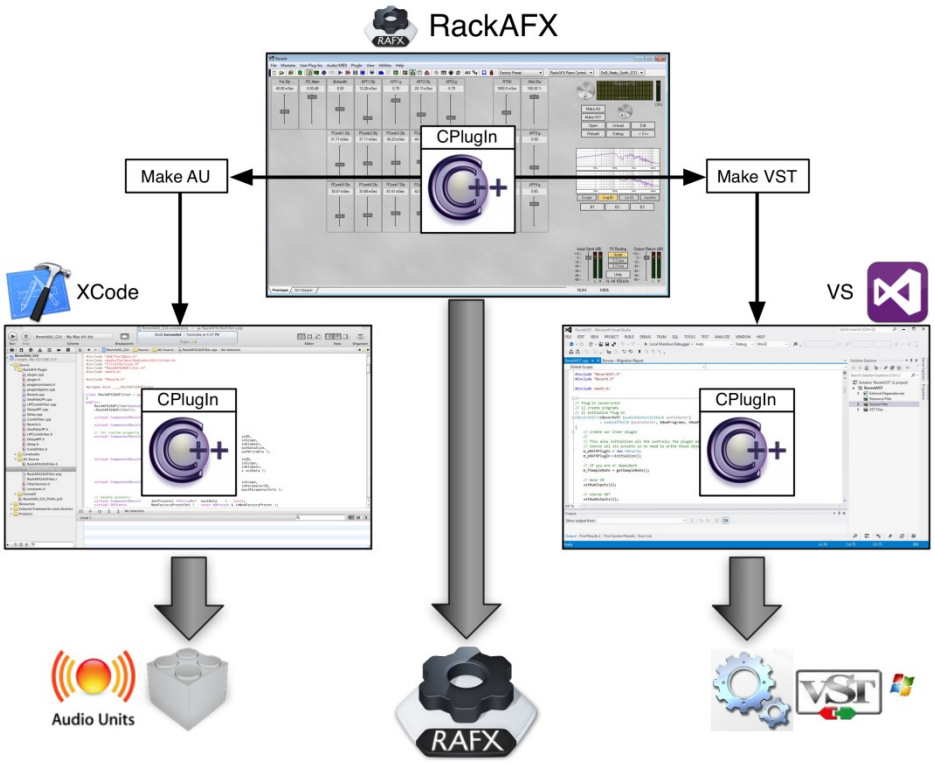
Je na platformě nezávislý framework postavený nad původním VST SDK. Obsahuje třídu `IPlug` řešící veškerou komunikaci s hostovací aplikací, třídu `IGraphic` obsluhující použité ovládací prvky a sadu tříd implementujících rozhraní `IControl`, které nabízejí interakci s uživatelem prostřednictvím grafického rozhraní. Ke kompilaci a komunikaci zásuvného modulu s hostovací aplikací je na platformě Windows zapotřebí přidat pouze dva soubory z VST SDK `aeffect.h` a `aeffectx.h`. Pro kompilaci do AU pro Mac OS X je potřeba získat Core audio SDK od firmy Apple. Tento projekt byl vypuštěn v roce 2007 v rámci http://forum.cockos.com/ fóra. Komunita není velká a poslední příspěvek týkající se tohoto frameworku byl zaslán v roce 2011. Nicméně obsahuje spousty užitečného kódu.

2.3.5 RackAFX

Tento projekt zkušeného programátora Willa Pirkla, který se programování zvukových aplikací věnuje mnoho let, je další možnou alternativou pro C++ vývoj VST zvukových zásuvných modulů. Jde o WYSIWYG editor usnadňující propojení kódu zpracovávajícího zvukový signál s příslušnými uživatelskými prvky grafického rozhraní. Tím například poskytuje programátorům vytvářející složité DSP algoritmy, rychlou cestu k jejich testování, aniž by museli být zpomalováni ruční implementací interakce modulu s jeho grafickým rozhraním. Zjednodušeně do editoru grafického rozhraní přetáhnete požadované grafické

ovládací prvky, kterým přiřadíte kusy kódu napsané v C++ jazyce. Je zde ale problém, který přináší každý WYSIWYG editor a to je skrytí mnoha stěžejních procesů na pozadí. To je pro začátečníky, zejména v tak komplexních technologiích jako je VST, velmi nepříjemné. Will Pirkl napsal také knihu [33] usnadňující programátorům proniknout do tajů tvorby zvukových zásuvných modulů pomocí RackAFX. Tak jako se například doporučuje před použitím WYSIWYG editorů pro tvorbu HTML stránek napsat několik webů v notepadu, tak i zde doporučuji napsat pár VST zásuvných modulů "ručně" a až po pochopení základních principů si práci usnadnit používáním RackAFX.

Web projektu [31] i samotný projekt byly nedávno aktualizovány o nové prvky, jako je export navrhnutého VST modulu do projektu, který může být otevřen ve Visual Studiu nebo XCode editoru. Poté už můžete pokračovat jako "opravdový" programátor. Bohužel já už jsem na konci své práce a novinky nestihnu prozkoumat. Těm, co by na mě chtěli navázat, doporučuji RackAFX znova prostudovat. Na obrázku 2.18 je nový princip fungování.



Obrázek 2.18: *Princip fungování RackAFX*

2.3.6 VST.NET

Je open-source projekt vydaný a udržovaný v rámci systému codeplex firmy Microsoft, který slouží k hostování open-source projektů. VST.NET se momentálně nachází ve verzi 1.0 a ta je označena jako stabilní. Jeho autor Marc Jacobi je velice aktivní a všechny mé dotazy byly obratem zodpovězeny. Projekt umožňuje vývojářům využít pro vytváření VST programů platformu .NET. Poskytuje vrstvu interoperabilty, která slouží jako most mezi COM

komponentami napsanými v nespravovaném kódu (C++) a základními stavebními prvky .NET programů - assemblies napsaných ve spravovaných jazycích .NET (VB, C#. VC++, atd.). To umožňuje použití COM komponent v .NET aplikacích a naopak. Využití této vlastnosti, je stěžejní v VST.NET, kde VST zásuvný modul napsaný ve spravovaném kódu potřebuje komunikovat se zvukovou pracovní stanicí napsanou v nespravovaném kódu. Zdrojové kódy VST.NET jsou velice dobře okomentovány a vstřícnost autora výborná. I přesto dá práci pochopit strukturu a procesy probíhající uvnitř frameworku. Důkazem může být i fakt, že projekt stáhlo přes 24 000 lidí, ale na internetu jsem nenašel jediný popis či návod, jak VST.NET použít. Z diskuze na codeplexu [10] vyplývá, že jen pár programátorů opravdu překonalo všechna úskalí a dokončilo svůj VST modul. Nevýhodou je podpora pouze OS Windows a možné problémy s rychlostí takto napsaného zásuvného modulu kvůli běhu pod Common language Runtime platformy .NET. Toto testuji v poslední kapitole této práce. Na druhou stranu velkým plusem je programování v mnohem příjemnějších jazycích jako jsou C# nebo VB, což zvyšuje rychlost vývoje a přináší možnost začlenění moderních .NET technologií pro vytváření grafického rozhraní (Winforms, WPF).

VST.NET obsahuje tři základní knihovny:

- Jacobi.Vst.Interop
- Jacobi.Vst.Core
- Jacobi.Vst.Framework

Jacobi.Vst.Interop

Tato část se stará o interoperabilitu mezi původním C a spravovaným kódem. Díky tomu, že C++ (VC++) je také jedním z jazyků .NET, je možné mixovat kód nespravovaného C++ jazyka se spravovaným VC++ v jedné assembly. Assembly `Jacobi.Vst.Interop` je přesně jednou z nich. Umožňuje volat nativní C rozhraní pomocí ukazatelů a opcodes, vytvořit z nativních struktur struktury spravované a směrovat veškerá volání na objekty. Komunikaci, směrování a překlad volání mezi hostovací aplikací a modulem zajišťují dva objekty. `PluginCommandStubProxy`(C++) s referencí na objekt `PluginCommandStub`, který implementuje C# rozhraní `IVstPluginCommandStub`. Proxy objekt bere volání a parametry z hostovací aplikace a převádí je do spravovaného Stub.

Jacobi.Vst.Core

Všechny C datové typy (většinou struktury a enumerátory) definované v původním VST mají svůj protějšek v C# `Jacobi.Vst.Core` assembly. Interop assembly tedy sahá do Core assembly aby byla schopna přecházet z nespravovaných typů do typů spravovaných. To je hlavní úkol Core assembly, avšak také obsahuje definici rozhraní `IVstPluginCommandStub`. Spravovaný VST zásuvný modul se dá programovat přímo oproti Core. Modul by implementoval `IVstPluginCommandStub`, což stačí pro komunikaci s hostovací aplikací. Zmíněné rozhraní však neřeší žádné strukturování, groupování, či rozdělení metod dostupných pro modul. Programování přímo oproti Core se dá využít pouze v situacích,

kdy máte již nějaký komplexní spravovaný kód pro zpracování zvuku a snažíte se jej adaptovat na platformu VST.

Jacobi.Vst.Framework

Tato část VST.NET je pro vývojáře nejzajímavější. Je to "nejvyšší" sada tříd a rozhraní pro vývoj VST modulu. Obzvláště důležité je rozhraní `IVstPlugin`, které musí každý zásuvný modul implementovat. `IVstPlugin` definuje základní komunikaci s hostovací aplikací. Poskytuje informace o jednotlivých dostupných funkcionalitách, autorovi, aktuální verzi modulu atd.

2.3.7 Napsaní vlastního frameworku

Pro komunikaci s hostovací aplikací je potřeba jen několika málo souborů z původního VST SDK. Tedy je možné tyto soubory vzít a nad nimi napsat celou logiku samostatně. To dělají všechny zmíněné frameworky. Jejich autoři jsou programátoři s mnohaletou praxi ve vývoji zvukových aplikací, VST zvukových zásuvných modulů a VST hostovacích aplikací. I jim vývoj frameworků a jejich uvedení do použitelného stavu zabralo několik let. Tuto variantu uvádím spíše jen pro úplnost.

2.4 Tvorba grafického rozhraní pro VST zvukové zásuvné moduly

Technologie Steinberg VST se s grafickým uživatelským rozhraním pro své zásuvné moduly vypořádává dvěma způsoby:

- Generické grafické rozhraní hostovací aplikace
- Zásuvný modul obsahuje své vlastní grafické rozhraní

2.4.1 Generické grafické rozhraní hostovací aplikace

Hostovací aplikace je schopna z kódu zásuvného modulu načíst informace o všech jeho parametrech a na základě toho vygenerovat generické uživatelské rozhraní automaticky. To sice šetří programátorovi spoustu času, ale také nedává prostor k použití svých grafických prvků. Tato varianta se hodí spíše ve fázi vývoje pro testování jednotlivých parametrů a algoritmů pro zpracování zvuku. Výhodou je, že takto napsaný zásuvný modul zůstane nezávislý na platformě (co se týče grafického rozhraní). Každá hostovací aplikace generuje grafické rozhraní po svém. Například Cubase používá pro zobrazení parametrů horizontální posuvníky obrázek 2.19, Orion Pro otočné knoby obrázek 2.20 a WaveLab simulaci přístrojů v racku obrázek 2.21. Na všech obrázcích se jedná o stejný zásuvný modul KillerRinger.

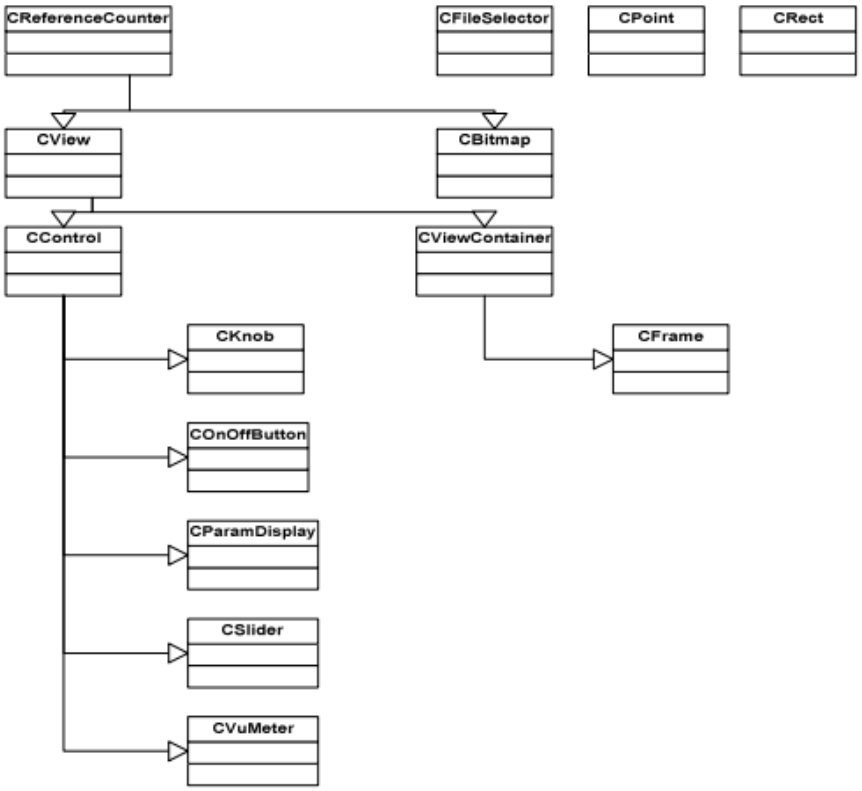
Obrázek 2.19: *Generické grafické rozhraní ve zvukové pracovní stanici Orion Pro*

Obrázek 2.20: *Generické grafické rozhraní ve zvukové pracovní stanici Cubase*

Obrázek 2.21: *Generické grafické rozhraní ve zvukové pracovní stanici WaveLab*

2.4.2 Vlastní implementace GUI

Jestliže požaduji osobitý vzhled a s vývojem VST zásuvného modulu to myslím opravdu vážně, musím přistoupit k variantě číslo dvě. Tou je vlastní implementace. Steinberg poskytuje již zmíněnou knihovnu VST GUI. Ta obsahuje třídu `AEffGUIEditor` pro implementaci grafického rozhraní a třídu `CControler` pro reakcí hodnot parametrů na změny provedené prostřednictvím grafických ovládacích prvků. VST GUI také disponuje řadou připravených základních druhů ovládacích prvků (obrázek 2.22). Jejich obdobu obsahují všechny výše zmíněné frameworky kromě VST.NET, kde si programátor musí vše udělat sám. Výhodou je, že internet je plný grafických prvků napsaných pro .NET, které jdou jednoduše integrovat do .NET projektu. RackAFX je WYSIWYG editor, kde je cesta k vlastnímu grafickému rozhraní jednoznačně nejrychlejší.



Obrázek 2.22: *Základní ovládací prvky grafického rozhraní ve Steinberg VST GUI*

3 Návrh VST zvukového zásuvného modulu

3.1 Volba nástrojů a prostředí k implementaci

Zvažování a testování všech uvedených cest k implementaci zásuvného modulu bylo opravdu zajímavé. Každá z nich má jisté výhody a nevýhody. Při mé volbě bylo rozhodující vyhnutí se nespravovanému C++ jazyku, možnost zapojení moderních nástrojů k vývoji softwaru a také celková hodnota mé práce pro ostatní studenty.

Jako nejméně probádaný jsem shledal VST.NET. Nejedná se o žádný WYSIWYG editor či novou vrstvu pro usnadnění vývoje nad původním Steinberg VST SDK. VST.NET pouze přemísťuje vývoj VST zásuvných modulů na platformu .NET. Je to sice o něco složitější cesta než s některými z výše zmíněnými frameworky, ale to považuji naopak za výzvu. Prozkoumání nové cesty k vývoji VST modulů a její zdokumentování bude sloužit studentům navazujícím na mou práci nejlépe.

VST.NET používá licenci GNU, která říká, že zkompilované knihovny mohou být použity jak pro nekomerční tak pro komerční účely. Když chce programátor dělat změny v zdrojovém kódu VST.NET, musí pak takto napsaný projekt vydat jako open-source. To přesně vyhovuje mým potřebám. Jako prostředí pro vývoj jsem zvolil Visual studio 2012, což se ostatně vzhledem k povaze VST.NET nabízí. Jako hostovací aplikaci pro testování mého zásuvného modulu použiji zvukovou pracovní stanici Steinberg Cubase 5 a také VSTHost aplikaci obsaženou ve VST.NET. Pro dotazy a řešení problému s vývojem diskuzi na domácích stránkách projektu[10].

3.2 Analýza požadavků

VST zvukový zásuvný modul bude poskytovat tuto funkcionalitu:

- Digitální zpracovaní vstupního signálu
- Poskytnutí zpracovaného signálu na výstupu
- Vystavení vlastních grafických prvků uživateli pro nastavení jednotlivých parametrů
- Korektní integraci do zvukové pracovní stanice

3.3 Volba potřebných rozhraní

Originální Steinberg VST SDK C++ část obsahuje třídy, které jsou postaveny nad C procedurálními rozhraními. Ty slouží jako bázové třídy, z kterých může zásuvný modul dědit. V VST SDK je struktura těchto bázových tříd svedena do jedné. Předpokládá se, že všechna funkcionalita zásuvného modulu bude implementována právě v jedné vyděděné třídě. Oproti tomu se VST.NET snaží jednotlivé funkcionality rozdělit a tím umožnit větší flexibilitu. Pro každou požadovanou funkcionalitu tedy poskytuje speciální rozhraní, které je implementováno zásuvným modulem a voláno vrstvou interoperability.

Podle analýzy požadavků v kapitole 3.2 potřebuji naimplementovat tyto rozhraní:

- IVstPlugin
- IVstAudioProcessor
- IVstBypass
- IVstPluginBuilder
- IVstPluginEditor

IVstPlugin

Toto rozhraní definuje metody `Open` a `Close` pro alokaci a uvolnění paměti potřebné pro zásuvný modul. Dále metody `Suspend` a `Resume` pro pozastavení a opětovného spuštění modulu. Nejdůležitější je zde Observable kolekce parametrů `Parameters`, která vystavuje a udržuje všechny parametry modulu.

IVstAudioProcessor

Díky implementaci tohoto rozhraní říkám, že můj zásuvný modul umí zpracovávat zvukový signál. Pro náročné se dá toto rozhraní zaměnit za `IVstAudioPrecissionProcessor`, který místo `float` typu obdrží do zvukového kanálu `double` (je přesnější). Rozhraní definuje dvě vlastnosti `BlockSize/SampleRate`. Některé hostovací aplikace pracují s `BlockSize` namísto `SampleRate`. Tyto vlastnosti jsou sice nastaveny automaticky hostovací aplikací, ale v algoritmech zpracování signálu se nimi často počítá a proto je vhodné vždy ošetřit obě vlastnosti. Další členem rozhraní je metoda `Process`, která poskytuje `IAudioChannel` kolekce (sady zvukových vzorků k zpracování) z hostovací aplikace zásuvnému modulu a zpátky.

IVstBypass

Není nepostradatelnou součástí mého modulu, ale tato funkcionalita patří k nejzákladnějším. Jedná se o možnost přerušit zpracování zvuku. Tedy ačkoli je efektový modul stále v efektovém racku, pracovní stanice ho ignoruje. To je důležitá funkce pro rychlé vypnutí vlivu efektu, což umožnuje okamžitě porovnat původní signál se zpracovaným.

IVstPluginEditor

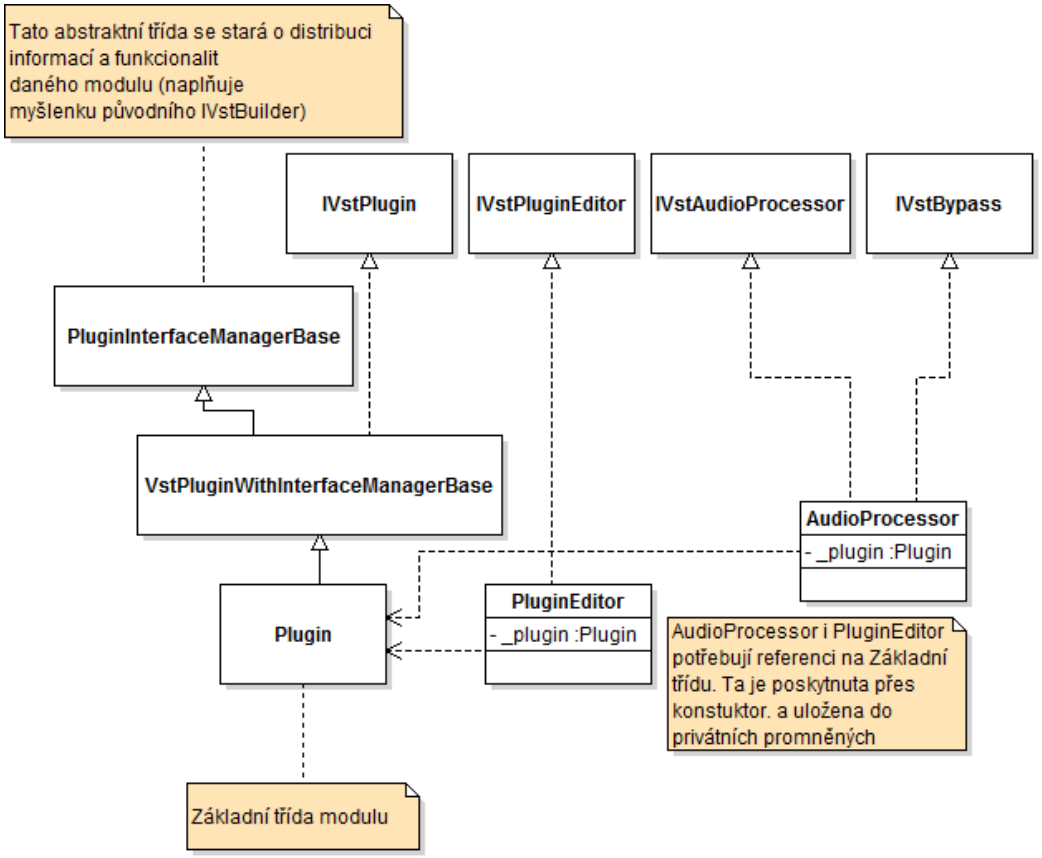
Pomocí tohoto rozhraní se vykresluje okno modulu. Důležitá je Metoda `Open`, kterou volá hostovací aplikace v momentě, kdy má být okno modulu otevřeno. Metoda `Close` zavře okno modulu.

IVstPluginBuilder

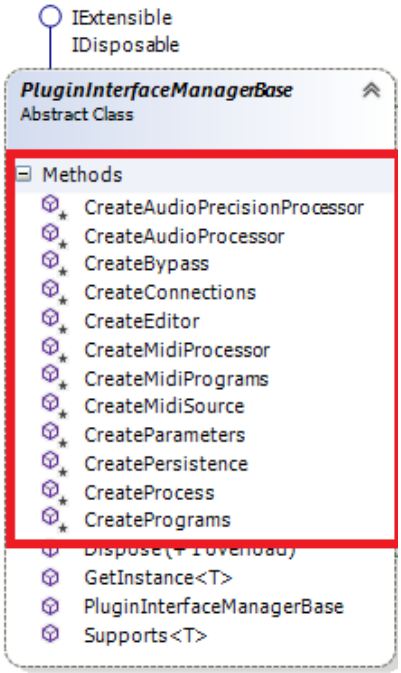
Mám-li sadu rozhraní, které definují dílčí funkcionality mého modulu, potřebuji tyto informace podat hostovací aplikaci a také na vyžádání vrátit referenci na danou implementaci. K tomu mělo sloužit rozhraní `IVstPluginBuilder`. Při konzultaci s tvůrcem frameworku jsem byl však upozorněn, že je tento mechanismus nakonec implementován pomocí abstraktní třídy `PluginInterfaceManagerBase`. Navíc framework VST.NET nabízí pokročilejší abstraktní třídu `VstPluginWithInterfaceManagerBase`, která implementuje jak

`PluginInterfaceManagerBase` tak `IVstPlugin`. Pro můj projekt jsem vybral druhou z nich.

Na obrázku 3.1 je zjednodušený diagram nejzákladnějších prvků potřebných pro implementaci mého VST zásuvného modulu. Ten má poskytovat zpracování zvukového signálu, vlastní grafické rozhraní a funkci Bypass. Po pochopení této základní struktury je již poměrně jednoduché přidat do vyvíjeného modulu další funkcionality. Pro jejich přidání je potřeba v základní třídě modulu přepsat příslušné zděděné metody CreateXXX(obrázek 3.2) bázové třídy `VstPluginWithInterfaceManagerBase`. Ty metody, které přepsané nejsou, vrací `null`. Na základě toho hostovací aplikace ví, že daná funkcionalita není podporována. Z obrázku se také dá jednoduše odvodit kompletní list všech možných funkcionalit.



Obrázek 3.1: *Schéma potřebných rozhraní*



Obrázek 3.2: *Seznam funkcionalit, které jsou dostupné pro zvukový modul*

3.4 Načtení modulu do hostovací aplikace

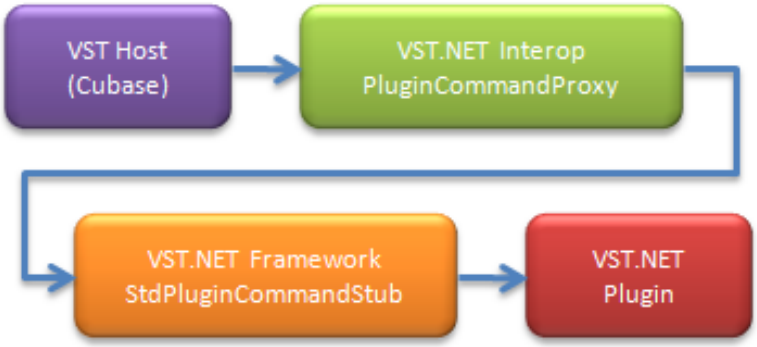
Velice důležitou součásti vývoje VST modulu pod VST.NET, je mechanismus jeho načtení do hostovací aplikace. Tedy jak Interop assembly (základní popis VST.NET assemblies se nachází v kapitole 2.3.6) získá referenci na implementaci IVstPluginCommandStub. V prostředí Windows hostovací aplikace očekávají VST moduly jako DLL knihovny. V případě vývoje pod originálním VST SDK jsou moduly kompilovány jako jediná nespravovaná DLL knihovna, která může být do hostovací aplikace jednoduše načtena. Kvůli míchání spravovaného a nespravovaného kódu je tento proces v VST.NET složitější.

Interop assembly exportuje `main` metodu. Právě tu hostovací aplikace očekává jako vstupní bod celé komunikace. Interop DLL je načtena do hostovací aplikace, se kterou si navzájem poskytnou ukazatele na funkce. Po jejich přijetí je Interop DLL nasměruje na funkce napsané spravovaným kódem. V podstatě si hostovací aplikace myslí, že právě interop DLL je daný modul. Ten je však ve skutečnosti ve spravované assembly zkompilovaného projektu VST modulu. Toto Interop assembly řeší za pomoci `Loaderu` umístěného v Core assembly, který vyhledá a načte spravovanou assembly VST modulu. Poté již může vytvořit instanci třídy implementující `IVstPluginCommandStub` rozhraní. Pro tento účel musí ve VST.NET existovat jmenná konvence. Tou je, že spravovaná assembly VST zásuvného modulu (C# kód implementující `IVstPluginCommandStub`) musí obsahovat postfix ".net". Jen tak může být nalezena a správně načtena. Například chci-li, aby se můj VST modul jmenoval 3TMatrixPlugin, je potřeba přejmenovat Interop assembly na 3TMatrixPlugin.dll a následně také samotnou assembly s kódem modulu na 3TMatrixPlugin.net.dll .

Veřejná třída implementující zmíněné rozhraní `IVstPluginCommandStub` je další povinou součástí každého VST modulu. Naštěstí VST.NET obsahuje abstraktní bázovou třídu `StdPluginCommandStub`, která toto rozhraní implementuje. Moje třída `PluginCommandStub` dědí právě z ní. Důležitou je zde abstraktní metoda `CreatePluginInstance`. Ta vrací základní objekt VST modulu implementující `IVstPlugin` rozhraní. Ve výpisu kódu 3.1 je implementace této třídy z mého projektu. Schéma na obrázku 3.3 ukazuje, jak celý proces probíhá od načtení Interop assembly hostovací aplikací až po finální vytvoření instance základní třídy VST modulu.

```
public class PluginCommandStub : StdPluginCommandStub
    {
        //Metoda vrací novou instanci základní třídy
        //VST modulu
        protected override IVstPlugin CreatePluginInstance()
        {
            return new Plugin();
        }
    }
```

Výpis kódu 3.1: *Ukázka metody PluginCommandStub*



Obrázek 3.3: *Životní cyklus VST Zdroj[32]*

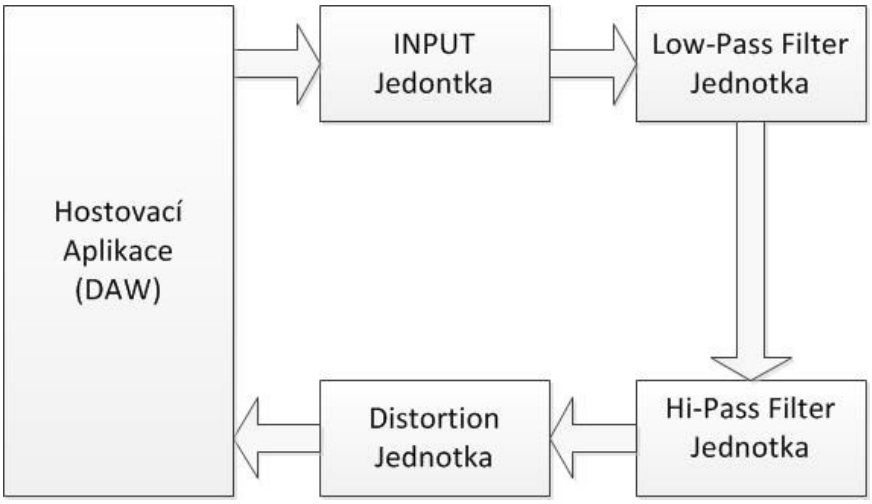
Pro správné načtení VST modulu je také velice důležité umístění DLL knihoven `Jacobi.VST.Core.dll` a `Jacobi.VST.Framework`. Ty by se měly nacházet ve stejném adresáři jako přejmenovaná Interop assembly a assembly VST modulu. V případě, že máte více VST modulů napsaných pomocí VST.NET, nabízí se instalace Core a Framework knihoven do Global Assembly Cache (CAG).

3.5 Návrh struktury VST modulu

Strukturu mého modulu a mnoho problému na které jsem během vývoje narazil jsem diskutoval na stránkách projektu VST.NET [27][10], kde na mé dotazy odpovídali prakticky pouze dva lidé. Nicméně jsme prodiskutovali spoustu témat a mělo by to být první místo, kam se čtenář mé práce zajímající se o vývoj zvukových modulů pomocí VST.NET měl podívat. Vzhledem k tomu, že tato práce je na naší univerzitě první svého druhu a můj primární záměr je přivést k programování VST modulů nové programátory, v průběhu jsem se rozhodl namísto vytváření složité struktury modulu a studia pokročilých DSP algoritmů, investovat většinu času do detailního pochopení VST.NET frameworku tak, abych vyprodukoval srozumitelnou bázovou práci pro navazující projekty. Nepopisuji zde tedy složité algoritmy, ale spíše demonstruji několik základních principů digitálního zpracování zvuku a jejich integraci do VST.NET frameworku.

Protože v DAW postrádám jednoduchý VST filtr, vybral jsem jako dvě jednotky modulu filtry. Dále modul bude poskytovat modulátory LFO a ADSR (více o nich bude psáno níže), které budou moci být přiřazeny různým zdrojům modulace. Poslední jednotka bude poskytovat efekt distorze.

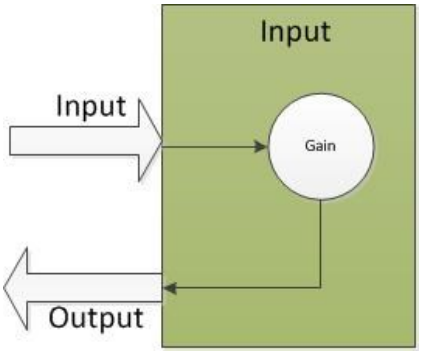
Výsledný VST modul bude v angličtině, proto již v návrhu budu pro jednotlivé parametry používat anglické označení.



Obrázek 3.4: *Základní schéma VST modulu 3TMatrixPlugin*

3.5.1 Návrh Input jednotky

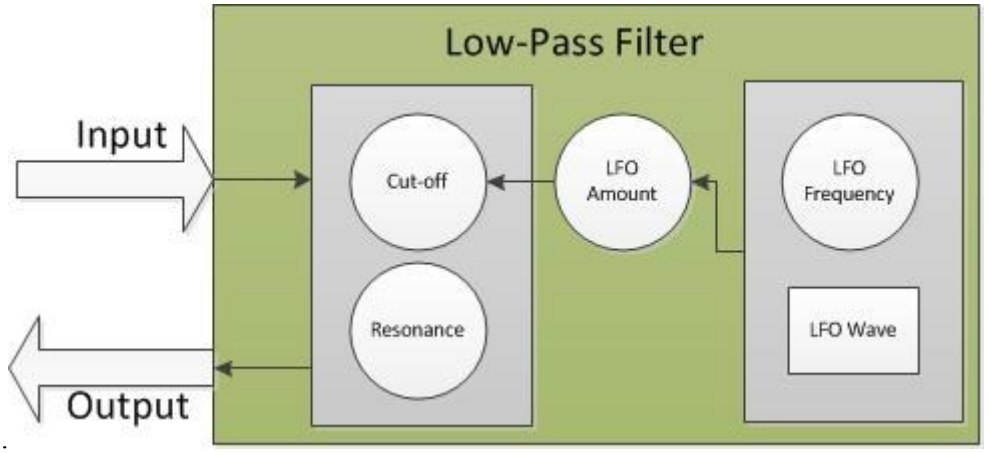
Tato jednotka bude sloužit jako vstupní bod zpracování signálů. V této verzi modulu bude obsahovat pouze jeden parametr umožňující regulaci vstupní hlasitosti signálu. Ten bude moci být modulován modulačními jednotkami. Hlavním důvodem zařazení je vize do budoucna, kdy bude moci být signál po vstupu do modulu nasměrován do libovolné jednotky. Uživatelské prvky umožňující tuto funkcionalitu budou umístěny právě zde.



Obrázek 3.5: *Návrh Input jednotky*

3.5.2 Návrh jednotky Low Pass Filter

Teorii k low-pass filtrům rozebírám v podkapitole 2.2.4. Hlavním úkolem Low pass Filter jednotky mého zásuvného modulu bude možnost ořezání vstupního signálu o horní frekvence v závislosti na uživatelem nastavené cut-off frekvence. Základní schéma můžete vidět na obrázku 3.6.

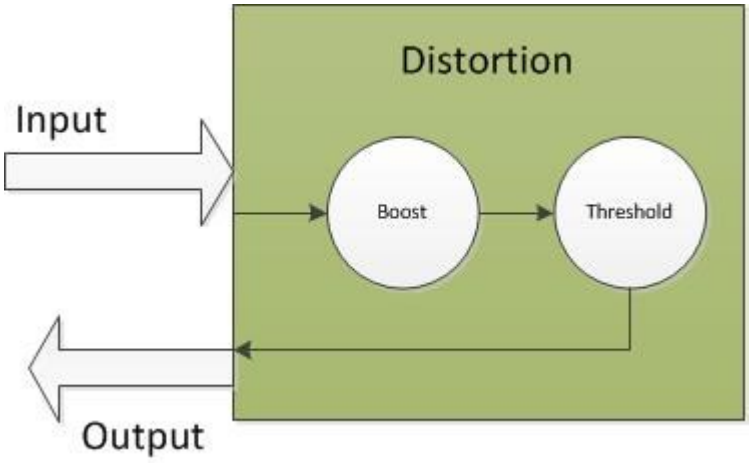


Obrázek 3.6: *Návrh Low-Pass Filter jednotky*

Volitelné bude přidání resonance (parametr Resonance) pro získání zajímavějšího výsledného zvuku. Po vzoru starých analogových syntetizérů bude sekce LFO napevno svázána s parametrem Cut-off. LFO Frequency umožní nastavení frekvence a LFO Wave typ generovaného modulačního signálu. O detailech LFO se dočtete v podkapitole 3.5.5 popisující modulační jednotku LFO1.

3.5.3 Návrh Distortion jednotky

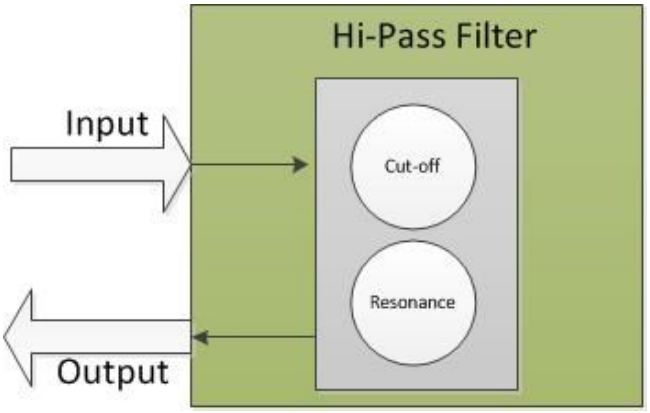
Pro nasimulování distorze jsem vybral techniku hard-clipping, kterou popisuji v podkapitolách 2.2.6 a 2.2.7. První z parametrů této jednotky bude Boost. Ten po vzoru kytarových distortion pedálu umožní uživateli předzesílit vstupní signál. Druhý parametr Threshold bude sloužit k změně hranice hard-clippingu.



Obrázek 3.7: *Návrh Distortion jednotky*

3.5.4 Návrh High Pass filter jednotky

Tato jednotka bude obsahovat hi-pass filtr pro ořezání spodních frekvencí signálu. Společně s parametrem pro nastavení cut-off frekvence bude obsahovat také parametr Resonance, který umožní přidání rezonance.

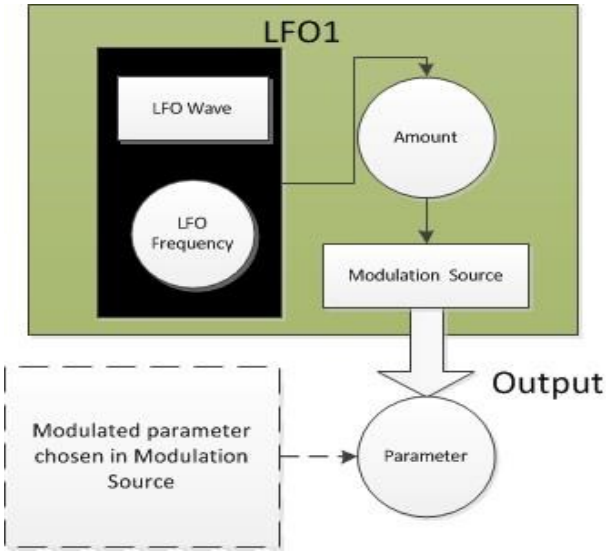


Obrázek 3.8: *Návrh Hi-Pass Filter jednotky*

3.5.5 Návrh Modulační jednotky LFO1

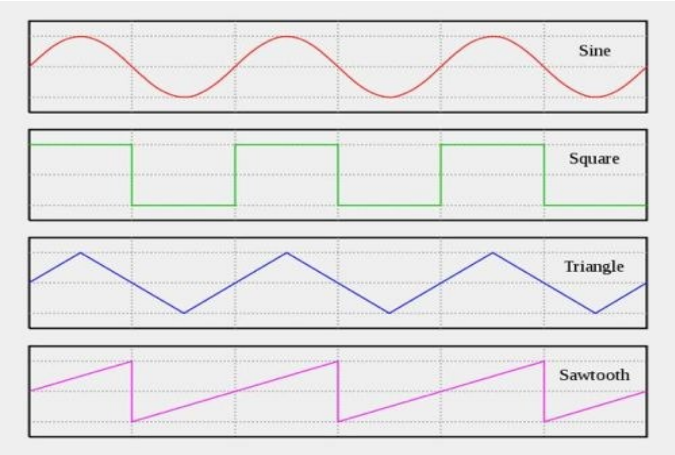
LFO(Low frequency oscillator) je generátor signálu obvykle s frekvencí pod 20 Hz. Takový signál vytváří rytmické pulsy. Často se používá k modulaci zvuků v syntetizérech nebo zvukových efektech. Aplikace modulace basové linky pomocí LFO je například běžnou technikou v moderním hudebním žánru dubstep. Ve zvukových efektech se například používá pro vytváření vibrata nebo tremola. Více informací o LFO naleznete zde [19] .

Jednotka mého VST modulu LFO1 umožní modulaci různých zdrojů modulace (parametrů). Jako vzor pro návrh této komponenty jsem použil klasické LFO jednotky syntetizérů. Ty obvykle poskytují parametry pro nastavení frekvence, tvaru generovaného signálu a míry ovlivnění zdroje modulace. Schéma návrhu je na obrázku 3.9.



Obrázek 3.9: *Návrh LFO1 jednotky*

Parametr LFO Wave bude umožňovat zvolit jeden ze čtyř základních typů generovaného signálu zobrazených na obrázku 3.10. K změně frekvence bude sloužit parametr LFO Frequency. Pomocí parametrů Amount a Modulation Source bude moci uživatel zvolit, co a jak modulovat.



Obrázek 3.10: *Čtyři základní typy vlny [20]*

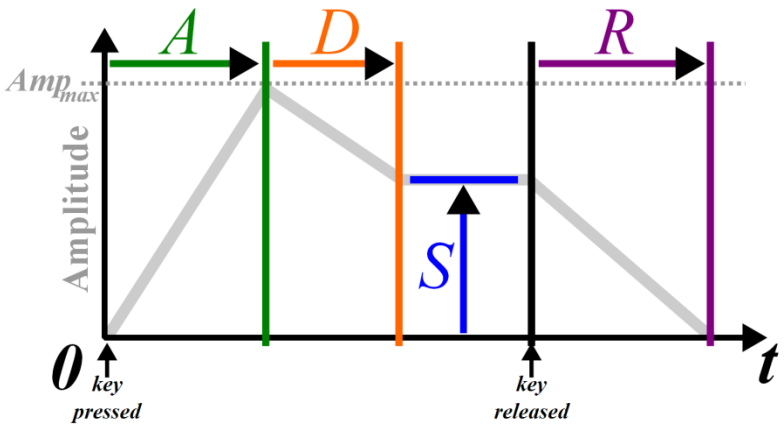
3.5.6 Návrh modulační jednotky ADSR Envelope

Druhým velice často používaným modulátorem ve zvukové technice je generátor ADSR obálky. Ta je definována čtyřmi parametry (stavy) :

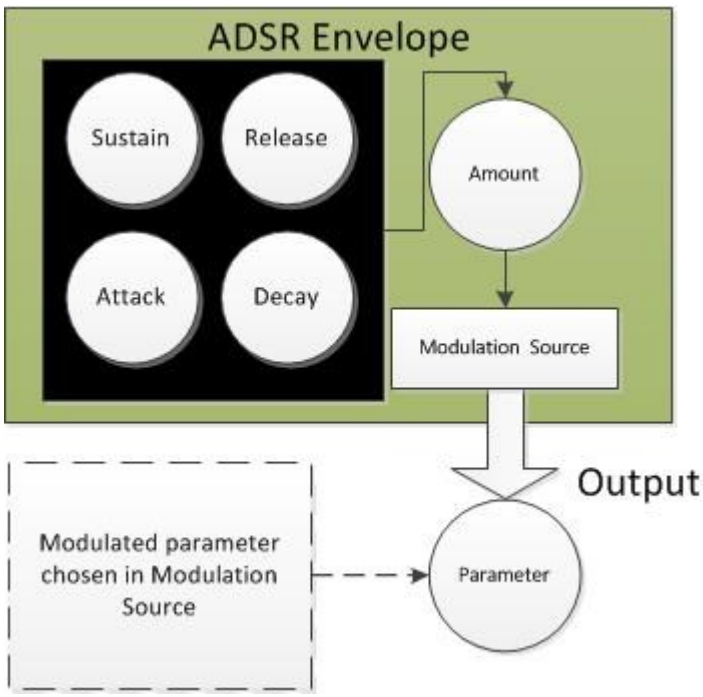
- **A**ttack time - výstup generátoru roste až do svého maxima, kde se přepne do stavu Decay
- **D**ecay time - výstup klesá až na hodnotu Sustain level, kde se přepne do stavu Sustain
- **S**ustain level - výstup zůstává na hodnotě Sustain level

- **R**elease time - výstup klesá z hodnoty Sustain level na nulu

V syntetizérech je generátor ADSR většinou aktivován stiskem klávesy. Poté zaktivuje svůj stav Attack, během kterého jeho výstup roste rychlostí závislé na nastavení Attack time parametru. Pokud klávesa zůstává stisknuta, výstup roste až do dosažení svého maxima. Je-li maximum dosaženo, přechází do stavu Decay a rychlostí závislou na parametru Decay time, klesá na uživatelem nastavený Sustain level. Když této hodnoty dosáhne, přepne svůj stav na Sustain a zůstane v něm až do uvolnění klávesy. Ve chvíli kdy je klávesa uvolněna, zaktivuje stav Release a výstup začne klesat rychlostí závislou na parametru Release time až k nule. Popsaný proces můžete vidět na obrázku 3.11.



Obrázek 3.11: *Princip ADSR obálky[18]*



Obrázek 3.12: *Návrh ADSR Envelope jednotky*

ADSR Envelope jednotka vygeneruje ADSR obálku závislou na nastavení Attack, Sustain, Decay a Release parametrů. Poté aplikuje zvolené množství modulace (parametr Amount) na vybraný zdroj modulace (parametr Modulation Source).

3.6 Návrh grafického rozhraní

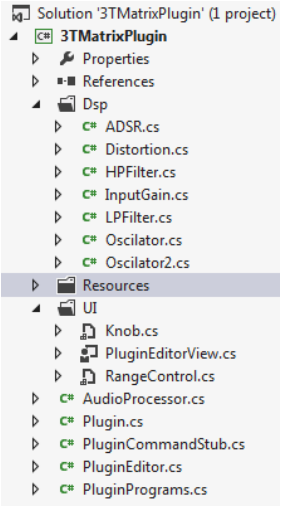
Grafické rozhraní bude vycházet z návrhů jednotlivých jednotek.

4 Implementace

V této části práce se věnuji implementaci VST modulu 3TMatrixPlugin podle návrhu provedeného v předchozí kapitole. Neuvádím matematický popis jednotlivých funkcí a algoritmů, protože pro implementaci mého modulu je důležitý hlavně jejich princip. Zajímá-li se čtenář o více matematický nebo technický popis, nalezne patřičné informace v přiložených referencích.

4.1 Struktura projektu

Jak jsem již zmínil, jako prostředí pro vývoj mého VST modulu jsem zvolil Microsoft Visual Studio. Pro lepší orientaci v projektu jsem vytvořil několik složek. Třídy starající se o zpracování signálu jsou umístěny ve složce DSP. Pro přidání grafických komponent, které nejsou originálně součástí Visual studia, slouží složka UI. Třídy implementující rozhraní vybrané ve fázi návrhu (podkapitola 3.3) jsou umístěny v kořenovém adresáři. Strukturu projektu můžete vidět na obrázku 4.1.



Obrázek 4.1: *Struktura VS projektu*

4.2 Implementace zpracování zvuku

Jádrem celého zpracování zvuku je třída `AudioProcessor.cs`. Ta implementuje rozhraní `IVstPluginAudioProcessor`, prostřednictvím kterého získává sady zvukových vzorků z hostovací aplikace. K jejich vyzvednutí slouží metoda `Process`. Zjednodušenou verzi této metody můžete vidět níže.

```
public override void Process(VstAudioBuffer[] inChannels,
VstAudioBuffer[] outChannels)
        {
            // nastavení input a outpu bufferů
            VstAudioBuffer[] _inChannels = inChannels;
            VstAudioBuffer[] _outChannels = outChannels;
```

```
            VstAudioBuffer input1 = _inChannels[0];
            VstAudioBuffer input2 = _inChannels[1];
            VstAudioBuffer output1 = _outChannels[0];
            VstAudioBuffer output2 = _outChannels[1];
            float sample = 0.0f;
            // if zde hlídá uživatelské nastavení Bypass funkce
            // v DAW
            if (!Bypass)
            {
                // prochází všechny přijaté vzorky
                for (int i = 0; i < input1.SampleCount; i++)
                {
                      // zpracování vzorku z prvního kanálu
                    sample = input1[i];
                    sample = InGain.ProcessSample(sample);
                    sample =
DistortionEffect.ProcessSample(sample);
                    sample = LpFilter.ProcessSample(sample);
                    sample = HpFilter.ProcessSample(sample);
                    output1[i] = sample;
                      // zpracování vzorku z druhého kanálu
                    sample = input2[i];
                    sample = InGain.ProcessSample(sample);
                         sample =
     DistortionEffect.ProcessSample(sample);
                    sample = LpFilter.ProcessSample(sample);
                    sample = HpFilter.ProcessSample(sample);
                    output2[i] = sample;
                }
            }
            else
            {
                base.Process(inChannels, outChannels);
      }
```

Výpis kódu 4.1: *Hlavní metoda pro zpracování zvuku Process*

Pomocí smyčky `for` prochází všechny kanály, posílá jejich vstupy jednotlivým DSP třídám a následně zpracovaný vzorek ukládá do výstupu, který obdrží zpátky hostovací aplikace. Také díky implementaci rozhraní `IVstPluginBypas` kontroluje stav proměnné `Bypass`, pro možnost dočasného vypnutí zpracování signálu.

Rozsahy parametrů zde neuvádím, protože pro pochopení většiny ukázek kódu nejsou důležité. Tam kde jsou, je uvádím. Pro případné zájemce o jejich hodnoty jsou lehko dohledatelné pomocí jejich definic v metodách `InitializeParameters` jednotlivých tříd. Zmínit zde však musím, že rozsahy pro uživatelské rozhraní nejsou totožné s rozsahy pro DSP výpočty. Uživatelské rozhraní pracuje s `int` hodnotami. Výpočty DSP algoritmů probíhají ve `float` hodnotách, protože rozsah hodnot zvukových vzorků z DAW je od -1 do1 včetně. Na pozadí tedy provádím konverzi mezi těmito dvěma rozsahy a typy. Hodnoty pro grafické prvky jsou uvedeny v uživatelské dokumentaci přiložené ke knihovnám modulu.

4.2.1 Implementace Low Pass filter jednotky

Implementace této jednotky je rozdělena do dvou tříd. `Oscilator.cs` kde je generován LFO signál a `LPFiler.cs` kde probíhá samotné filtrování. Při inicializaci každá z tříd volá metodu `InitializeParametrs`. Ta se stará o nadefinování parametrů, které jsou později propojeny s grafickými prvky modulu a slouží k ovlivňování DSP algoritmů. Ukázku takové definice můžete vidět níže. Jedná se o parametr pro ovládání LFO frekvence.

```
private void InitializeParameters()
        {
           //kolekce všech parametrů modulu
            VstParameterInfoCollection parameterInfos =
plugin.PluginPrograms.ParameterInfos;
           //vytvoření nové definice parametru
            VstParameterInfo paramInfo = new VstParameterInfo();
           //LFOFrequency param
            paramInfo = new VstParameterInfo();
            paramInfo.CanBeAutomated = true;
            paramInfo.Name = "LFOfreq";
            paramInfo.Label = "LFO Frequency";
            paramInfo.ShortLabel = "%";
            //je možno pracovat jak s int tak s float
            // tvůrcem VST.Net mi však bylo doporučeno
             // použití intu, zde je prostor k testování
            //které jsem už nestihl
            paramInfo.MinInteger = 1;
            paramInfo.MaxInteger = 20;
            paramInfo.LargeStepInteger = 1;
            paramInfo.StepInteger = 1;
            paramInfo.DefaultValue = 1.0f;
            LFOFrequency = new VstParameterManager(paramInfo);
            VstParameterNormalizationInfo.AttachTo(paramInfo);
            // přídání do kolekce definicí
            parameterInfos.Add(paramInfo);
        }
```

Výpis kódu 4.2: *Definice parametru LFO Frequency*

Na konci definice jsou informace o parametru přidány do kolekce `ParameterInfos`, která je definovaná v základní třídě modulu `Plugin.cs` a slouží pro sdílení parametrů napříč všemi třídami.

Algoritmus pro filtrování jsem si vypůjčil od Paula Kelleta [21]. Filtr se skládá ze dvou sériových low-pass filtrů prvního řádu. Díky tomu poskytuje strmost -12dB/octave. Vytváří také zpětnou vazbu díky, které je generována také rezonance.

Vestavěný LFO může modulovat cut-off frekvenci. K aplikaci této modulace slouží modulační proměnná `cutoffMod`. Modulační proměnné jsou mnou zvolené označení proměnných ovlivňujících míru modulace v jednotlivých jednotkách. Jejich princip bude jasnější po přečtení další podkapitoly 4.2.2.

Ve výpisu kódu 4.3 je algoritmus low-pass filtru. Vzorky přicházejí do metody `ProcessSample`. Do výpočtu filtrace vstupují proměnné `_calculatedCutoff` a `feedbackAmount`, které jsou závislé na uživatelském nastavení parametrů a spočítány příslušnými metodami `getCalculatedCutoff` a `calculateFeedbackAmount`.

```
        float getCalculatedCutoff()
        {
            // Výpočet cut-off frekvence
            //proměnná cutoff je propojená
            //s knobem v uživatelském rozhraní
            //cutoffMod odráží míru modulace vestavěným LFO
            float c =
Convert.ToSingle(Math.Max(Convert.ToSingle(Math.Min(cutoff +
cutoffMod, 0.99)), 0.01));
            return c;
        }
        private void calculateFeedbackAmount()
        {
            //proměnná resonance drží hodnotu uživatelského
            //parametru Resonance
            //Výpočet je závislý na cut-off frekvenci
            feedbackAmount = resonance + resonance / (1.0f -
getCalculatedCutoff());
        }
public float ProcessSample(float inputValue)
        {
           // Vrací Cut-off + modulační proměné
            float _calculatedCutoff = getCalculatedCutoff();
//prní filter společně se zpětnou vazbou spočítanou
//nazákladě Resonance
            buf0 +=  calculatedCutoff * (inputValue - buf0 +
feedbackAmount * (buf0 - buf1));
           //druhý filtr
            buf1 += _calculatedCutoff * (buf0 - buf1);
           // vrátí zfiltrovaný vzorek
             return buf1;
        }
```

Výpis kódu 4.3: *Algoritmus Low-Pass filtru*

4.2.2 Implementace jednotky ADSR Envelope

Tato jednotka je naprogramována ve třídě ADSR.cs. Třídy, které obsahují parametry dostupné k modulaci touto obálkou, obsahují proměnné pojmenované jmény odvozenými z názvu zdroje modulace a modulátoru. Typickým zástupcem může být modulační proměnná `cutoffEnvelopeMod` ve třídě `HPFilter`.cs. Její hodnota se pak přičítá k hodnotě parametru Cut-off. Nelze modulovat přímo parametry, protože jsou připojeny k prvkům grafického rozhraní. Jednotlivé DSP třídy si také hlídají hodnotu parametru Amount, kterou se jednotlivé modulační proměnné násobí. Tento parametr má rozsah pro uživatele od -100% do +100%. Pro výpočty na pozadí se však dělí 100. Výchozí hodnota je nula. Po vynásobení

modulační proměnné nulou je také modulace nulová. V momentě kdy chce uživatel uplatnit modulaci, změní hodnotu parametru Amount, která se stane nulovou a proto začne zvyšovat nebo snižovat hodnotu modulační proměnné.

Čí modulační proměnná (potažmo parametr) má být ovlivňována řídí pomocí parametru Modulation Source struktura `switch` umístěná v metodě `Process` třídy `AudioProcessor.cs`. Tento princip je také uplatněn v modulaci pomocí LFO1 jednotky.

Při implementaci této komponenty jsem vycházel z popisu ADSR obálek zde [22] a blogu věnujícímu se vývoji VST modulů pomocí originálního Steinberg SDK [23].

Samotný algoritmus generování se skládá ze tří částí. První řeší exponenciální průběh změny hlasitosti. Lidský sluch vnímá změnu hlasitosti logaritmicky. Aby generovaná změna byla vnímána lineárně, je třeba ji provádět exponenciálně. Nabízí se použití funkce `Math.Exp`. Ta je ale výpočetně náročná. Existuje mnohem chytřejší způsob tohoto výpočtu viz [26]. O exponenciální průběh se stará proměnná `multiplier`. Její výpočet je ve výpisu kódu

```
void calculateMultiplier(double startLevel, double endLevel,
long lengthInSamples)
        {
            multiplier = Convert.ToSingle(1.0 +
(Math.Log(endLevel) - Math.Log(startLevel)) /
(lengthInSamples));
        }
```

Výpis kódu 4.4: *Výpočet proměnné multiplier*

Druhou částí ADSR generátoru je metoda `enterStage`, která při zavolání nastavuje výchozí hodnoty pro započetí nového stavu. Proměnná `nextStageSampleIndex` udává délku stavu nastavenou uživatelem. Proměnná `currentSampleIndex` hlídá během generování signálu aktuální pozici v daném stavu a proto zde musí být vždy resetována.

```
public void enterStage(EnvelopeStage newStage)
        {
            currentStage = newStage;
            // resetuj pozici stavu na začátek
            currentSampleIndex = 0;
            if (currentStage == EnvelopeStage.ENVELOPE_STAGE_OFF
||
                currentStage ==
EnvelopeStage.ENVELOPE_STAGE_SUSTAIN)
            {
                // během těchto stavu se neděje nic
                nextStageSampleIndex = 0;
            }
            else
            {
                /*za kolik vzorků znova volat enterStage*/
                nextStageSampleIndex =
Convert.ToInt64(getStageValue(currentStage) * sampleRate);
```

```
            }
            /* podle toho v jaké fázi se nachází
               nastaví multifier a currentLevel*/
            switch (newStage)
            {
                    // v tomto stavu generuj nulu
                case EnvelopeStage.ENVELOPE_STAGE_OFF:
                    currentLevel = 0.0f;
                    multiplier = 1.0f;
                    break;
                    /*v Attack stavu jdi z nuly do maxima
                      bude to trvat nextStageSamplIndex
                      vzorků*/
                case EnvelopeStage.ENVELOPE_STAGE_ATTACK:
                    currentLevel = minimumLevel;
                    calculateMultiplier(currentLevel,
                                        1.0,
                                        nextStageSampleIndex);
                    break;
                /*jdi z maxima na hodnotu Sustain
                  za nextStageSamplIndex vzorků*/
                case EnvelopeStage.ENVELOPE_STAGE_DECAY:
                    currentLevel = 1.0f;
                    calculateMultiplier(currentLevel,

Math.Max(getStageValue(EnvelopeStage.ENVELOPE_STAGE_SUSTAIN),
minimumLevel),
                                        nextStageSampleIndex);
                    break;
                    //nastav aktuální hodnotu na Sustain LVL
                case EnvelopeStage.ENVELOPE_STAGE_SUSTAIN:
                    currentLevel =
getStageValue(EnvelopeStage.ENVELOPE_STAGE_SUSTAIN);
                    multiplier = 1.0f;
                    break;
                    //vrať se zpátky na nulu za dobu
                case EnvelopeStage.ENVELOPE_STAGE_RELEASE:
                     calculateMultiplier(currentLevel,
                                        minimumLevel,
                                        nextStageSampleIndex);
                    break;
                default:
                    break;
            }
        }
```

Výpis kódu 4.5: *Metoda enterStage starající se o stavy ADSR generátoru*

Ve chvíli kdy jsou připraveny změny stavů a proměnné umožňující exponenciální průběh obálky mohu volat metodu `nextSample` generující obálku. Ta při každém zavolání

inkrementuje proměnnou `currentSampleIndex` a na základě porovnání s `nextStageSampleIndex` rozhoduje, pro který stav bude generovat.

```
public float nextSample()
        {
            EnvelopeStage newStage;
            if (currentStage != EnvelopeStage.ENVELOPE_STAGE_OFF
&&
                currentStage !=
EnvelopeStage.ENVELOPE_STAGE_SUSTAIN)
            {
                //jestliže si dosáhl dalšího stavu přepni stav
                if (currentSampleIndex == nextStageSampleIndex)
                {
                 if (currentStage ==
EnvelopeStage.ENVELOPE_STAGE_RELEASE)
                    {
                        newStage =
EnvelopeStage.ENVELOPE_STAGE_OFF;
                    }
                    else
                    {
                        newStage = currentStage + 1;
                    }
                    enterStage(newStage);
                }
                //vynásob generevonaný signál
                //pro dosažení exp. průběhu
                currentLevel *= multiplier;
                // zvyš aktuální pozici
                currentSampleIndex++;
            }
            return (float)currentLevel;
        }
```

Výpis kódu 4.6: *Metoda nextSample pro generování ADSR obálky*

To by velice dobře fungovalo v případě propojení s MIDI událostmi. Ty by byly nastaveny, jako spoušť generování. Tedy přechodu ze stavu OFF do stavu ATTACK. Také by udávaly délku stavu SUSTAIN. Během studia a testování VST.NET jsem bohužel zjistil, že VST efekt nemůže přijímat MIDI informace. Proto bylo třeba ještě přidat kus kódu do metody, která ADSR generátor volá `AudioProcessor.Process` Ten můžete vidět na výpisu kódu níže. Jedná se o periodickou spoušť generování ADSR obálky.

```
…
// jestliže je obálka ve stavu OFF zapni stav ATTACK
if (Envelope.CurrentStage ==
ADSR.EnvelopeStage.ENVELOPE_STAGE_OFF)
Envelope.enterStage(ADSR.EnvelopeStage.ENVELOPE_STAGE_ATTACK);
// jestliže je obálka ve stavu SUSTAIN zapni RELEASE
```

```
                    if (Envelope.CurrentStage ==
ADSR.EnvelopeStage.ENVELOPE_STAGE_SUSTAIN)
Envelope.enterStage(ADSR.EnvelopeStage.ENVELOPE_STAGE_RELEASE);
…
```

Výpis kódu 4.7: *Periodická spoušť ADSR generátoru*

ADSR generátor by se dal naprogramovat i mnohem jednodušeji, avšak v této formě je připraven pro rychlou integraci do VST instrumentů, které již MIDI protokol podporují.

4.2.3 Implementace jednotky LFO1

LFO1 generátor umožňuje uživateli nastavení frekvence pomocí parametru LFO Frequency. Ten slouží k určení posunu v čase `phaseIncrement`(výpis kódu 4.8). Výpočet také ovlivňuje vzorkovací frekvence nastavená v DAW `sampleRate`, protože posun v čase počítám ve vzorcích.

```
private void updateIncrement()
        {
            // nastevnení kroku v čase
            phaseIncrement = (frequency * 2 * PI) / (
sampleRate);
        }
```

Výpis kódu 4.8: *Nastavení kroku v čase pro LFO*

Samotné generování LFO probíhá v metodě `nextSample`, kde struktura `switch` rozhoduje na základě nastavení parametru LFO Wave, který tvar signálu má být generován. Princip generování sinusového průběhu můžete vidět ve výpisu kódu 4.9. Pro ostatní průběhy se algoritmus liší pouze v řádku nastavujícím `_generatedSample`, podle požadovaného tvaru.

```
public float nextSample()
        {
            float _generatedSample;
            double twoPI = 2 * mPI;
            // podle zvoleného parametru LFO Wave
            switch (mOscillatorMode)
            {
                case OscillatorMode.OSCILLATOR_MODE_SINE:
                    //generátor sinusového průběhu
                    //lfoPhase hlídá aktuální pozici
                    //sinusioidy
_generatedSample = (float)Math.Sin(lfoPhase);
                    //posuň se v čase o phaseIncrement
                    lfoPhase += phaseIncrement;
                    //když je dosáhnuta perioda
                    //začni znova
                    while (lfoPhase >= twoPI)
                    {
                        lfoPhase -= twoPI;
                    }
                    return _generatedSample;
```

```
                case OscillatorMode.OSCILLATOR_MODE_SAW:
                     //obdobně i pro jiné průběhy
                …
            }
            return 0;
        }
```

Výpis kódu 4.9: *Kód generující LFO*

4.2.4 Implementace High Pass Filter jednotky

Princip implementace této jednotky je kombinací principů zmíněných výše. Protože může být modulována jak LFO1 jednotkou tak ADSR Envelope jednotkou, a je možno modulovat jak Cut-off tak Resonance, bylo třeba vytvořit čtyři modulační proměnné.

Na rozdíl od dvou filtrů prvního řádu v Low Pass filter jednotce obsahuje filtry čtyři. Z toho vyplývá větší strmost filtrace -24dB/octave. Viz výpis kódu 4.10.

```
//modulační proměnné
       private float cutoffEnvelopeMod;
        private float resonanceEnvelopeMod;
        private float cutoffLFO1Mod;
        private float resonanceLFO1Mod;
//cut-off frekvence závisí na parametru Cut-off propojeného
//s cutoff //proměnnou. Taky ale na cuoffEnvelopeMod a
//cutoffLFO1Mod z modulátorů
float getCalculatedCutoff()
        {
            float c =
Convert.ToSingle(Math.Max(Convert.ToSingle(Math.Min(cutoff +
cutoffEnvelopeMod + cutoffLFO1Mod, 0.99)), 0.01));
            return c;
        }
//množství resonance závisí na parametru Resonance propojeného
//s resonance proměnnou. Taky ale na resonanceEnvelopeMod a
//resonanceLFO1Mod z modulátorů
        float getCalculatedResonance()
        {
            float c =
Convert.ToSingle(Math.Max(Convert.ToSingle(Math.Min(resonance +
resonanceEnvelopeMod + resonanceLFO1Mod, 0.99)), 0.0));
            return c;
        }
public float ProcessSample(float inputValue)
        {
            float calculatedCutoff = getCalculatedCutoff();
// čtyři hi-pass filtry v sérii
            buf0 = buf0 + calculatedCutoff * (inputValue - buf0
+ feedbackAmount * (buf0 - buf1));
            buf1 = buf1 + calculatedCutoff * (buf0 - buf1);
            buf2 += calculatedCutoff * (buf1 - buf2);
```

```
            buf3 += calculatedCutoff * (buf2 - buf3);
            return inputValue - buf3; ;
        }
```

Výpis kódu 4.10: *Implementace hi-pass filtru*

4.2.5 Implementace Distortion jednotky

Pro implementaci distorze jsem zvolil užití hard-clippingu. Předzesílení slabých signálů obstarává parametr Boost. Při jeho nejnižší hodnotě je vstupnímu signálu ponechána původní hodnota. Při nejvyšší je zesílen stokrát. Již toto způsobuje v mnoha případech clipping. Parametr Threshold slouží pro nastavení hranice clippingu ještě níže. Čím níže je nastaven, tím více distorze přináší. Kvůli snižování Threshold, se však také snižuje amplituda výstupního signálu. To řeším na konci zpracování tím, že signál dělím hodnotou Threshold, což slouží jako kompenzace hlasitosti. Celý algoritmus je ukázán ve výpisu kódu 4.11.

```
public float ProcessSampleWithPreAmp(float sample)
        {
            //hodnoty parametrů nastavené uživatelem
            float _Threshold =
            this.DistThreshold.CurrentValue/  100;
            float _distBoost = this.DistBoost.CurrentValue;
            float _sample = sample;
            //je-li signál kladný nebo nula
            if (sample >= 0)
            {
                //předzesílení signálu
                _sample = _sample * (1 + _distBoost);
                //v případě překročení Threshold vrátí
                //její hodnotu
                // Hard-clipping shora
                _sample = Math.Min(_sample, _Threshold);
                //kompenzace ztráty hlasitosti
                _sample /= _Threshold;
                return _sample;
            }
            else
            {
                //Je-li signál záporný
                _sample = _sample * (1 + _distBoost);
                // Hard-clipping zezdola
                _sample = Math.Max(_sample, -_Threshold);
                _sample /= _Threshold;
                return _sample;
            }
        }
```

Výpis kódu 4.11: *Algoritmus pro hard-clipping distorzi*

4.3 Implementace Parametrů

Hostovací aplikace může reagovat na akce uživatele prostřednictvím parametrů. V VST.NET reprezentuje každý parametr třída `VstParameterManager`, která obsahuje definici parametru třídy `VstParameterInfo`. Každá z mých tříd starajících se o DSP definuje své parametry. Jejich nadefinování je umístěno v metodách `InitializeParameters`(), které jsou volány při vytváření DSP objektů. Ukázka tohoto procesu je ve výpisu kódu 4.12.

```
class LPFilter
    {
internal VstParameterManager LPFilterCutOff { get; private set;
}
        internal VstParameterManager LPFilterResonance { get;
private set; }
        internal VstParameterManager LPFilterLFOAmount { get;
private set; }
public LPFilter(Plugin plugin)
        {
…
            InitializeParameters();
…
        }
private void InitializeParameters()
        {
            VstParameterInfoCollection parameterInfos =
_plugin.PluginPrograms.ParameterInfos;
            VstParameterInfo paramInfo = new VstParameterInfo();
            //LPResonance
            paramInfo = new VstParameterInfo();
            paramInfo.Category = paramCategory;
            paramInfo.CanBeAutomated = true;
            paramInfo.Name = "LPres";
            paramInfo.Label = "LPResonance";
            paramInfo.ShortLabel = "%";
            paramInfo.MinInteger = 1;
            paramInfo.MaxInteger = 100;
            paramInfo.LargeStepInteger = 1;
            paramInfo.StepInteger = 5;
            paramInfo.DefaultValue = 1.0f;
            LPFilterResonance = new
VstParameterManager(paramInfo);
            VstParameterNormalizationInfo.AttachTo(paramInfo);
            parameterInfos.Add(paramInfo);
…
//definice dalších parametrů
        }
```

Výpis kódu 4.12: *Princip vytváření parametrů*

To by stačilo pro informování hostovací aplikace o dostupných parametrech. V případě implementace vlastního grafického rozhraní je však třeba udělat další kroky popsané v následující podkapitole.

4.4 Implementace grafického rozhraní

Hostovací aplikace už ví o parametrech pomocí Observable kolekce `ParameterInfos`. Jednotlivé `VstParameterInfo` instance jsou propojeny s `VstParameterManager` instancemi obsluhujícími aktuální hodnoty parametru. Pro interakci s grafickým rozhraním zbývá `VstParameterManager` podat třídě implementující okno modulu. Tou je v mém projektu `PluginEditorView.cs`. Propojení se odehrává ve třídě `PluginEditor.cs`, která implementuje rozhraní `IVstPluginEditor`, kvůli kterému musí obsahovat metodu `Open` pro otevření okna. V této metodě tedy vytvářím kolekci `VstParameterManager` instancí a posílám ji mému oknu. To je zobrazeno v zjednodušeném výpisu kódu třídy `PluginEditor.cs` níže.

```
namespace _3TMatrixPlugin
{
    internal sealed class PluginEditor : IVstPluginEditor
    {
        private Plugin _plugin;
     //Wrapper pro Windows form okna
     // ve VST.NET je také funkční wrapper pro WPF
        private WinFormsControlWrapper<PluginEditorView> _view;

        public PluginEditor(Plugin plugin)
        {
            _plugin = plugin;
            _view = new
WinFormsControlWrapper<PluginEditorView>();
        }

        public void Open(IntPtr hWnd)
        {
            // vytvoření kolekce managerů pro okno modulu
            var paramList = new List<VstParameterManager>()
                {
_plugin.AudioProcessor.InGain.InGain,
_plugin.AudioProcessor.DistortionFX.DistThreshold,
_plugin.AudioProcessor.DistortionFX.DistMakeUp,
_plugin.AudioProcessor.DistortionFX.DistMakeUpOnOff,
_plugin.AudioProcessor.LpFilter.LPFilterCutOff,
_plugin.AudioProcessor.LpFilter.LPFilterResonance,

   …
                //další parametry
                };
```

```
                // podá seznam parametrů oknu modulu
            _view.SafeInstance.InitializeParameters(paramList,
_plugin);
            //otevře okno
            _view.Open(hWnd);
        }
…
}
```

Výpis kódu 4.13: Podání parametrů oknu modulu

Okno prohlížeče už je klasická WinForms komponenta, která obdrží v metodě `InitializeParameters` seznam parametrů. Poté tento seznam procházím a jednotlivé parametry propojuji s patřičnými ovládacími prvky grafického rozhraní. Za zmínku stojí problém vláken, na který jsem zde narazil. Protože parametry jsou spravovány a upravovány vláknem starajícím se o zpracování zvuku a grafické rozhraní vláknem jiným, může zde docházet ke kolizím. Například když automatizace hostovací aplikace mění hodnotu grafického prvku. Proto je třeba při křížení vláken používat `Invoke`. Detaily o tomto tématu najdete v diskuzi týkající se vývoje mého modulu zde [27]. Důležitá je zde také proměnná `executeEvent`, která hlídá, aby nedošlo k nekonečné smyčce při reakci na události. To je také diskutováno v [27].

```
public partial class PluginEditorView : UserControl
    {
        public PluginEditorView()
        {
            InitializeComponent();
        }
        Plugin _plugin;
       //hlídá aby nedošlo k nekonečné smyčcce při reakcina
       //události
        private bool executeEvent = false;
        internal bool
InitializeParameters(List<VstParameterManager> parameters,
Plugin plugin)
        {
            _plugin = plugin;
            if (parameters == null || parameters.Count < 1)
return false;
            // připoj GainParameter Gain knobu
            BindParameterToKnob(parameters[0], InputGainLabel,
InputGainKnob, InputGainShortLabel, InputGainValueLabel);
           //připoj DistortionThreshold parametr knobu
            BindParameterToKnob(parameters[1],
DistortioThresholdLabel, DistortionThresholdKnob,
DistortioThresholdShortLabel, DistortioThresholdValueLabel);
// dále se volají Bindovací metody pro různé typy ovládacích
//prvků např. BindParameterToComboBox atd… Až do posledního
//prvku v kolekci
            executeEvent = true;
```

```
            return true;
        }
// ukázka jedné  Bind metody, parametr si vymění hodnoty s
//ovládacím prvkem a zaregistrují se patřičné události pro
//změny v GUI a pro změny příchozí zevnitř Modulu.
private void BindParameterToKnob(VstParameterManager paramMgr,
Label label, Knob knob, Label shortLabel, Label ValueLabel)
        {
            label.Text = paramMgr.ParameterInfo.Name;
            shortLabel.Text = paramMgr.ParameterInfo.ShortLabel;
            knob.Minimum = paramMgr.ParameterInfo.MinInteger;
            knob.Maximum = paramMgr.ParameterInfo.MaxInteger;
            knob.Value = Convert.ToInt32(paramMgr.CurrentValue);
            ValueLabel.Text = paramMgr.CurrentValue.ToString();
            knob.Tag = paramMgr;
            paramMgr.PropertyChanged += new
PropertyChangedEventHandler(_knobVstParameterManager_PropertyCha
nged);
            knob.ValueChanged += new
System.EventHandler(Knob_ValueChanged);
        }
…
// EventHandlery a další Bind metody
}
```

Výpis kódu 4.14: *Propojení parametrů s grafickými prvky*

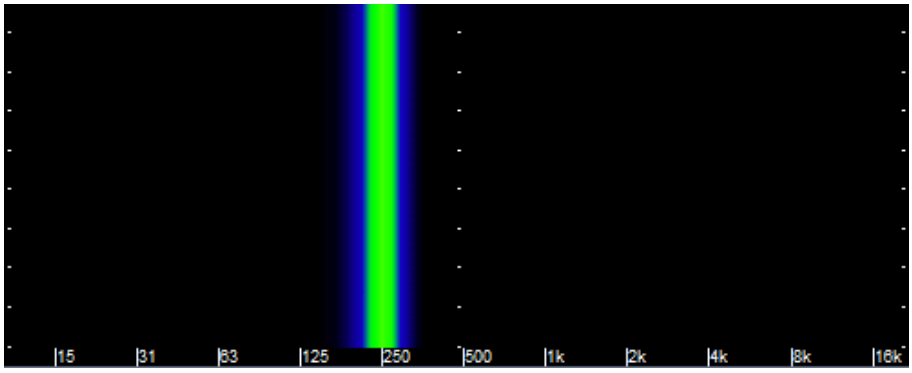
Pro použití Windows Presentation Foundation (WPF) stačí ve třídě `PluginEditor.cs` nahradit `WinFormsControlWrapper` za `WpfControlWrapper`, který naleznete ve staženém balíku VST.NET u příkladu CorePlugin.

Výsledné grafické rozhraní mého VST modulu můžete vidět v další kapitole na obrázku 5.6.

5 Ověření funkčnosti a zhodnocení výsledků

5.1 Testování DSP

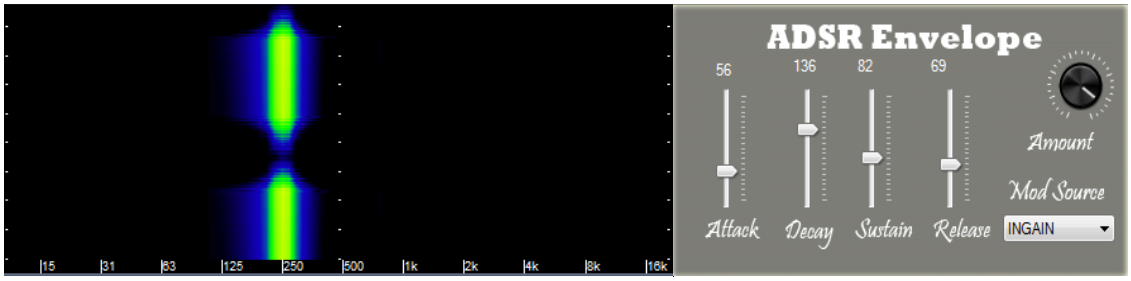
Pro testování zpracování zvuku jsem využíval hlavně svůj sluch. Ten však zdokumentovat nemůžu, proto jsem dále použil VST analyzér IXL Spectrum Analyzer firmy Roger Nichols. Ten poskytuje vizualizaci frekvenčního spektra signálu v čase. Pro všechny testy používám jednoduchý sinusový vstupní signál s frekvencí okolo 250 Hz, který je vidět na obrázku. Čas plyne zezdola nahoru.



Obrázek 5.1: *Testovací vstupní signál*

5.1.1 Test ADSR generátoru

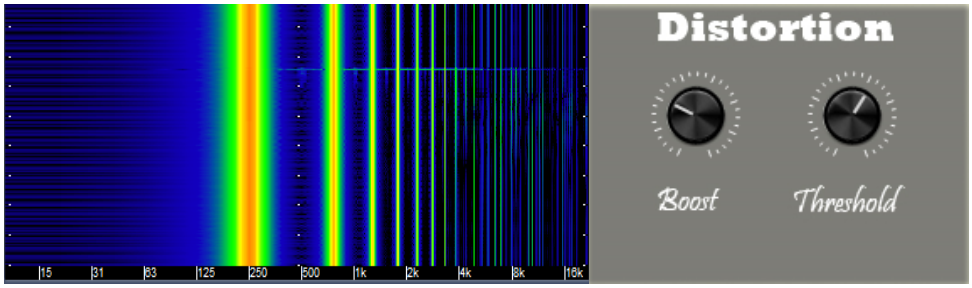
Jako zdroj modulace ADSR generátoru jsem nastavil InGain parametr, který ovlivňuje hlasitost signálu. Poté zvýšil množství modulace pomocí parametru Amount. Na obrázku 5.2 je vidět, jak generovaná obálka moduluje vstupní signál ve všech svých stavech, které jsou nastaveny parametry Attack, Sustain, Decay a Release.



Obrázek 5.2: *Test funkčnosti ADSR jednotky*

5.1.2 Test Distortion jednotky

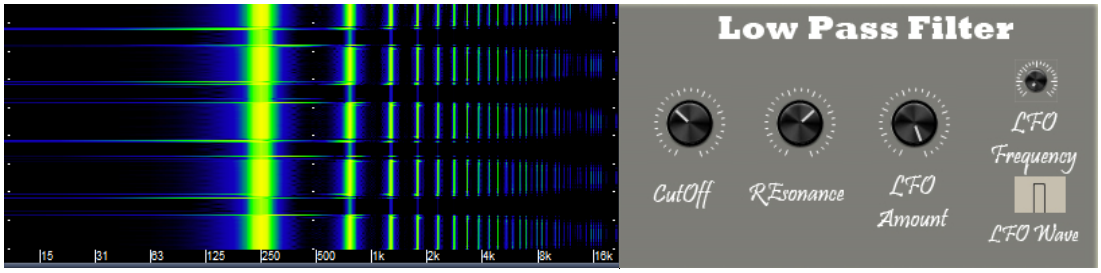
Při distorzi by měly vznikat vyšší harmonické frekvence. Pomocí parametru Boost jsem signál zesílil. Tento parametr umožnuje zesílení až stokrát. V momentě kdy signál překročí hlasitost nastavenou parametrem Threshold, dochází ke clippingu (ořezání), který má za následek vznik výše zmíněných vyšších harmonických frekvencí. Ty jsou vidět na obrázku 5.3.



Obrázek 5.3: *Test funkčnosti ADSR Envelope jednotky*

5.1.3 Test Low Pass filter jednotky

Tato jednotka disponuje svým vlastním LFO generátorem, který je propojen s Cutt-off parametrem. Funkčnost je nejlépe vidět při nastavení frekvence LFO na 1 Hz. Navýšení hodnoty parametru Amount má za následek aplikaci LFO modulace Cut-Off parametru. Zvýšení Resonance přidává zvuku barvu. To jde ale nejlépe poznat sluchem. Test ověřuje funkčnost jak LFO generátoru této jednotky tak filtrování signálu závislé na parametru Cut-off. Jeho pozice se mění v závislosti na LFO, pro který byl vybrán tvar Square. Výsledný vliv na signál demonstruje obrázek 5.4.

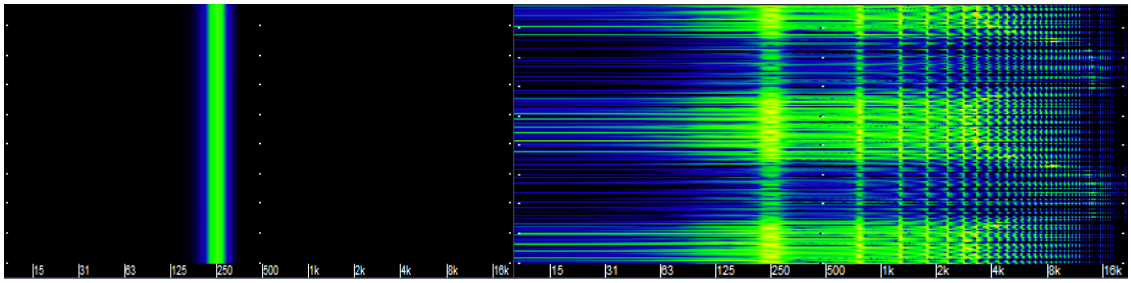


Obrázek 5.4: *Test funkčnosti Low Pass filter jednotky*

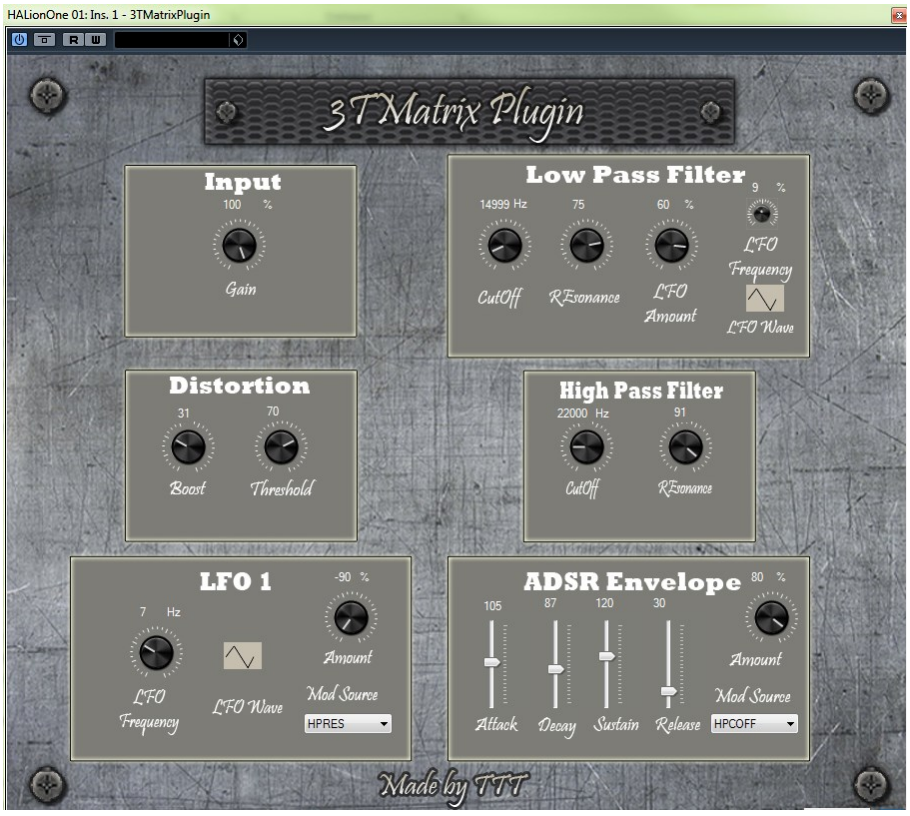
5.1.4 Komplexní test všech jednotek najednou

V tomto testu jsou otestovány zbylé jednotky LFO1 a High Pass Filter. Dále se zde testuje možnost zapojení všech jednotek najednou.

Na vstupu zůstává stále jednoduchý sinusový signál z obrázku 5.1. Ten je nejprve zpracován Distortion jednotkou, která mu přidá vyšší harmonické frekvence. V Low Pass filter jednotce jsou odfiltrovány ty nejvyšší z nich a následně vstupuje do High-Pass jednotky. Její Cut-off parametr je modulován ADSR obálkou. LFO1 jednotka moduluje parametr Resonance. Nastavení všech parametrů můžete vidět na obrázku 5.6. Výsledkem celého procesu je výstupní signál zobrazený na obrázku 5.5. Tento test demonstruje možnosti přetvoření původního zvuku téměř k nepoznání, což bylo primárním požadavkem na vyvíjený modul. To dává kreativcům do rukou mocný nástroj a je jen na nich, jaký zvuk pomocí tohoto VST modulu vytvoří.



Obrázek 5.5: *Srovnání signálu před a po zpracování*



Obrázek 5.6: *Nastavení parametrů pro test všech jednotek najednou*

5.2 Testování grafického rozhraní

Nejlepším testerem grafického rozhraní aplikace je koncový uživatel. Proto jsem požádal několik kolegů, kteří VST moduly používají k tvorbě vlastních skladeb, aby můj produkt nahráli do svých zvukových stanic a otestovali jeho funkčnost. Pro jejich zpětnou vazbu jsem vytvořil krátký formulář. K jednotlivým bodům odpovídali Ano/Ne nebo hodnotili stupnicí od 1 (nejhorší) do 10 (nejlepší). Na konci dostali prostor k vlastním komentářům a návrhům pro zlepšení. Zodpovězené formuláře jsou přiloženy na CD.

V komentářích si nejvíce stěžovali na nepopsané rozsahy hodnot jednotlivých ovládacích prvků. Protože mám domluvený grafický design od kamaráda grafika, bude toto brzy vyřešeno. Momentálně probíhá diskuze možných vzhledů. Jeden z tázaných si také

stěžoval na rozházenost grafických prvků po načtení modulu do DAW. Nikde jinde jsem, ale toto chování nepozoroval. Bude třeba vystopovat, co je příčinou. Shrnutí jejich odpovědí ukazuje tabulka 5.1.

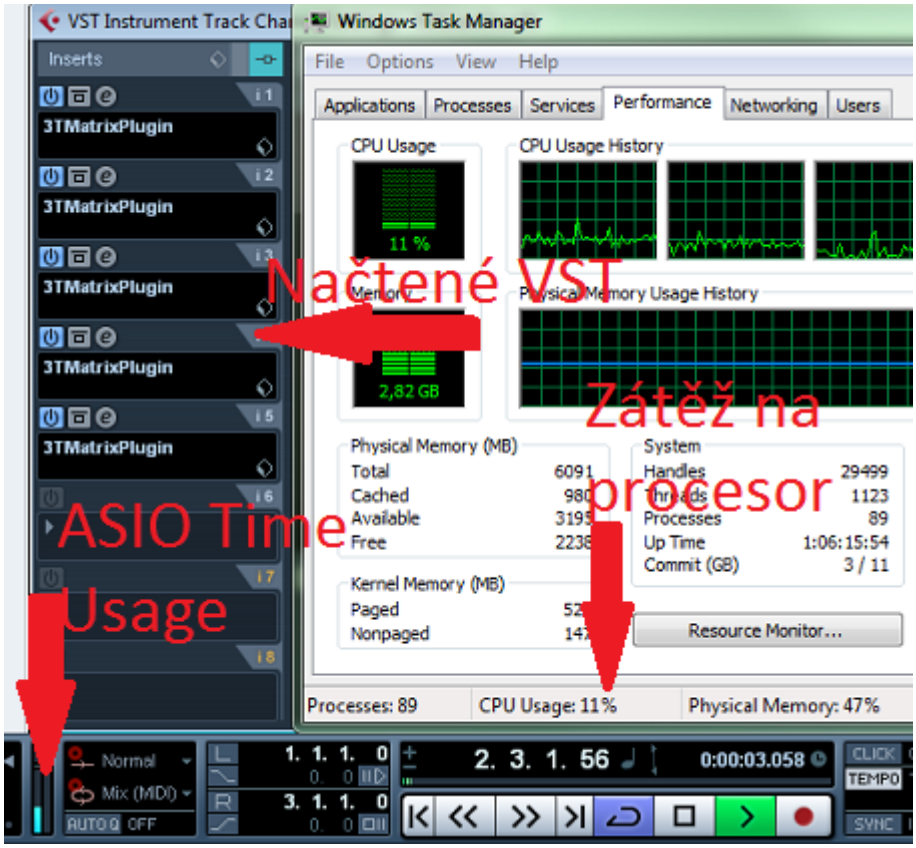
Tabulka 5.1 Shrnutí výsledků dotazníku

	DaMos23	**Bingoy**	**Ludwigo**	**UrvanBlue**
Použitý DAW	Reaper	Cubase 5	FL studio 10	Cubase 5.1
Podařilo se modul načít do DAW	ANO	ANO	ANO	ANO
Přehlednost jednotlivých ovládacích prvků	7	8	7	6
Očekávaný vliv na zvuk při změnách hodnot	6	8	7	7
Kvalita DSP	8	6	5	7
Originalita výsledného zvuku	6	6	7	6
Poskytnuté možnosti pro kreativitu	7	8	6	8
Stojí za to ve vývoji pokračovat	ANO	ANO	ANO	ANO

5.3 Testování výkonu

Testování výkonu a zátěže na hostovací systém proběhl pomocí načtení pěti instancí modulu do Inserts racku DAW Cubase 5. Všechny byly aktivní a zpracovávaly procházející signál. Zátěž CPU se po aktivaci modulů zvedla pouze o dvě procenta z (9% na 11%). To je na VST moduly příjemný výsledek, protože mnoho z nich již při několika instancích přesáhnou možnosti procesoru.

O něco hůře dopadlo měření ASIO Time Usage. Pět běžících instancí se sice stále drží v hranicích přípustnosti, ale při velkých projektech by mohly nastat problémy. Toto je pravděpodobně způsobeno velkou výpočetní náročností někde v DSP algoritmech. Bude třeba vystopovat toto slabé místo a porozhlédnout se po alternativách.



Obrázek 5.7: *Test výkonu v prostředí DAW Cubase 5*

5.4 Shrnutí dosažených výsledků

Dosažené výsledky testování jsou spíše pozitivní. Vzhledem k tomu že je tento VST modul mým prvním, jsem s nimi spokojen. Všechny jednotky fungují a zátěž na hostovací systém je v mezích přípustnosti. Také zpětná vazba od uživatelů byla obstojná.

Závěr

V této práci jsem se věnoval technologii Virtual Studio Technology vytvořené firmou Steinberg pro rozšiřitelnost počítačových zvukových pracovních stanic. Práce začíná historickým rozborem vývoje elektronické hudební techniky a plynule přechází k pojmu zvukového zásuvného modulu. Protože jsem informace o zvukové technice získal převážně samostudiem, obsah teoretické části byl průběžně konzultován s vedoucím práce a také s několika členy internetového fóra audiozone.cz, kteří mají s touto problematikou mnoho let praktických zkušeností. Tímto jsem se snažil předejít případným chybám. Obdržel jsem také několik velice pozitivních reakcí ohledně zajímavosti a zpracování práce.

V druhé části se již blíže věnuji technologii VST. Popisuji různé typy VST zásuvných modulů, jejich účel a základní principy procesů, které v nich probíhají. Také uvádím výsledky mého průzkumu možností jejich implementace. Diskutuji vhodnost jednotlivých možností pro konkrétní potřeby programátorů.

Po teoretické části začíná část praktická. Nejprve provádím analýzu požadavků a poté spolu s analýzou VST.NET frameworku návrh jednotlivých komponent systému. Z toho vychází následující kapitola věnující se implementaci VST zásuvného modulu 3TMatrixPlugin. Rozebírám implementaci každé z komponent a společně uvádím okomentované ukázky z kódu pro vysvětlení jednotlivých algoritmů. Poslední část práce se věnuje testování modulu ve zvukové pracovní stanici Cubase 5 pomocí IXL Spectrum analyzéru. Jednotlivé testy jsou popsány a jejich výsledky demonstrovány pomocí obrázků s výstupy IXL Spectrum analýzeru. Zdokumentovaný je také test zatížení hostovacího systému. Další testování proběhlo za účasti několika hudebníků, které jsem požádal o vyzkoušení modulu na jejich stanicích. Skrze krátký dotazník odpovídali na otázky týkající se grafického rozhraní a celkové funkčnosti modulu.

Během programování jsem prošel mnoha bezradnými momenty. Nakonec se mi však podařilo zmíněný VST modul dokončit tak, aby vyhovoval zadání práce. Další verze modulu bude obsahovat možnost změny směrování signálu mezi jednotlivými komponentami a také poskytne pokročilejší sekci pro správu zdrojů modulace a jejich modulátorů. Kolega momentálně vytváří nový grafický design, se kterým modul bude působit mnohem profesionálněji. Dále je třeba doladit jednotlivé parametry a ovladače tak, aby co nejlépe sloužily potřebám uživatelů. Otevřenou kapitolou jsou také DSP algoritmy, které se určitě dají vylepšit nebo nahradit lepšími. Tedy navazující práce je spousta a rozhodně nehodlám po odevzdání práce s vývojem VST modulů skončit. Lidí věnujících se v České republice programování VST modulů je jen několik, ačkoli jejich uživatelů máme přinejmenším stovky. Proto jsem se snažil práci psát tak, aby její přečtení přilákalo nové programátory. Ti sice nestojí na podiích před tisícími diváky, ale bez jejich práce by velká část hudební produkce, kterou známe z dnešních obrovských festivalů, nikdy nevznikla. Jsem otevřený jakékoli spolupráci, takže mě v případě zájmu o programování VST modulů neváhejte kontaktovat.

Použitá literatura

[1] MARKUS SCHNEIDER. Česko-německý magazín - elektronická hudba - retrospektiva. Online-Redaktion, GoetheInstitut e.V. [online].2006 [cit. 2014-2-14]. Dostupné z <http://www.goethe.de/ins/cz/pra/kul/duc/emu/ges/cs1575400.htm>

[2] FRANTIŠEK KOPECKÝ. ejazz.cz - Historie elektronické taneční hudby [online]. [cit. 2011-01-05]. Dostupné z <http://www.ejazz.cz/articles/elektroakusticka-hudba-sedesata-leta/ >

[3] Wikipedia.org - Digital audio workstation [online]. [cit. 2014-2-30]. Dostupné z <http://en.wikipedia.org/wiki/Digital_audio_workstation>

[4] Wikipedia.org - Musical Instrument Digital Interface [online]. [cit. 2014-2-30]. Dostupné z <http://cs.wikipedia.org/wiki/Musical_Instrument_Digital_Interface>

[5] Wikipedia.org - Korg M1 [online]. [cit. 2014-2-30]. Dostupné z <http://en.wikipedia.org/wiki/Korg_M1>

[6] Wikipedia.org - Virtualization [online]. [cit. 2014-3-30]. Dostupné z <http://en.wikipedia.org/wiki/Virtualization>

[7] Wikipedia.org - Virtual Studio Technology [online]. [cit. 2014-3-28]. Dostupné z <http://en.wikipedia.org/wiki/Virtual_Studio_Technology>

[8] Dith.it - dokumentace Steinberg VST SDK 2.4 [online]. [cit. 2014-4-28]. Dostupné z <http://www.dith.it/listing/vst_stuff/vstsdk2.4/doc/html/index.html>

[9] MARC JACOBI. Vstnet.codeplex.com - domácí stránka projektu VST.NET [online]. [cit. 2014-3-30] Dostupné z <https://vstnet.codeplex.com/>

[10] MARC JACOBI. Vstnet.codeplex.com - discussions [online]. [cit. 2014-4-30] Dostupné z <https://vstnet.codeplex.com/discussions/>

[11] Wikipedia.org - Audio Filter [online]. [cit. 2014-4-15] Dostupné z <http://en.wikipedia.org/wiki/Audio_filter>

[12] DAVID BAZALA. Amapro.cz - filtry [online]. [cit. 2014-4-15] Dostupné z <http://amapro.cz/public/ele/filtry.php>

[13] PAUL WHITE. Soundonsound.com - Equalisers explained [online]. [cit. 2014-4-3] Dostupné z <http://www.soundonsound.com/sos/jul01/articles/equalisers1.asp>

[14] MARK GARRISON. Music.tutsplus.com - Encyclopedia of Home Recording: Equalizer [online]. [cit. 2014-4-15] Dostupné z <http://music.tutsplus.com/articles/encyclopedia-of-home-recording-equalizer--audio-12573>

[15] Wikipedia.org - Dynamic range compression [online]. [cit. 2014-4-15] Dostupné z <http://en.wikipedia.org/wiki/Dynamic_range_compression>

[16] JIŘÍ ROKOVSKÝ. Avmainia.cz - Loudness war: válka, kterou prohrává posluchač [online]. [cit. 2014-4-12]. Dostupné z <http://avmania.e15.cz/loudness-war-valka-kterou-prohrava-posluchac-ukazka>

[17] PAUL WHITE. Soundonsound.com - Compression & Limiting[online]. [cit. 2014-3-17]Dostupné z http://www.soundonsound.com/sos/1996_articles/apr96/compression.html>

[18] En.wikiaudio.org – ADSR envelope [online]. [cit. 2014-16-4] Dostupné z <http://en.wikiaudio.org/ADSR_envelope>

[19] Wikipedia.org - Low frequency oscillation [online]. [cit. 2014-16-4] Dostupné z <http://en.wikipedia.org/wiki/Low-frequency_oscillation>

[20] Sheponbass.co.uk - Signal CHain Bassics, Part 3: Modulation [online]. [cit. 2014-16-4] Dostupné z <http://sheponbass.co.uk/blog/signal-chain-bassics-part-iii-modulation.html>

[21] Musicdsp.org – Resonant filter [online]. [cit. 2014-16-4] Dostupné z <http://www.musicdsp.org/showone.php?id=29>

[22] Earlevele.com – Envelope generators – ADSR Part 1 [online] [cit. 2014-16-4] Dostupné z <http://www.earlevel.com/main/2013/06/01/envelope-generators/>

[23] Martin-finke.de – Making Audio Plugins Part 11: Envelopes [online] [cit. 2014-16-4] Dostupné z <http://martin-finke.de/blog/articles/audio-plugins-011-envelopes/>

[24] Wikipedia.org – Distortion (music) [online] [cit. 2014-16-4] Dostupné z <http://en.wikipedia.org/wiki/Distortion_(music)>

[25] GUERIN ROBERT. Velká kniha MIDI: Brno: Computer Press, 2004. ISBN 8072269852

[26] Christian Schoenebeck. Musicdsp.org – Fast Exponencial Envelope Generator [online]. [cit. 2014-15-4]. Dostupné z <http://www.musicdsp.org/showone.php?id=189>

[27] Vstnet.codeplex.com – The best Plugin Structure for my Plugin and all about developing VSTMatrixPlugin [online] [cit. 2014-25-4]. Dostupné z <https://vstnet.codeplex.com/discussions/541925>

[28] Test-nastroju.webnode.cz – Zvukové efekty a jejich stručný popis [online]. [cit. 2014-25-4] Dostupné z <http://test-nastroju.webnode.cz/nahravani/efekty-a-jejich-strucny-popis/>

[29] Juce.com – Domácí stránka projektu JUCE [online].[cit. 2014-25-4] Dostupné z <http://www.juce.com/about-juce>

[30] Jvstwrapper.sourceforge.net – Domácí stránka projektu jVSTwRapper [online].[cit. 2014-25-4] Dostupné z <http://jvstwrapper.sourceforge.net/>

[31] Willpirkle.com – RackAFX [online]. [cit. 2014-25-4] Dostupné z <http://www.willpirkle.com/rackafx/>

[32] Obiwanjacobi.blogspot.cz – VST.NET Plugin Structure [cit. 2014-25-4]. Dostupné z <http://obiwanjacobi.blogspot.cz/2008/06/vstnet-plugin-structure.html>

[33] WILL PIRKLE. Designing Audio Effect Plug-Ins in C++. Burlington: Focal Press, 2013. 560 s. ISBN 9780240825151

Seznam příloh

Součástí BP/DP je CD/DVD.

Adresářová struktura přiloženého CD/DVD:

Dotazníky

Text diplomové práce

Uživatelská příručka 3TMatrixPlugin

VST modul 3TMatrixPlugin

Zdrojové kódy